Open Peer Review on Qeios

RESEARCH ARTICLE

How to Build an IoT System with AI Models to Predict Forest Fires in California

Jun Wang[1]

1 Arizona State University

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.

Abstract

In this weather monitoring system, you will learn how to use the DHT22 and BMP180 sensors to collect temperature, humidity, pressure, and altitude data. This data will be sent to a MySQL database for storage via AWS MQTT, allowing for email notifications to users and enabling future research. The Raspberry Pi 4 is utilized in this project as a compact and portable computer to connect the DHT22 and BMP180 sensors. The Raspberry Pi 4 will have AWS-IOT-MQTT installed to transmit data from the Pi to a Restful server. The Restful server, built with FastAPI, will persist the data into the MySQL database and send email notifications to users.

This guide will demonstrate the proper use of sensors to construct your IoT system. The cost of the project can be influenced by the price of different sensors, and using reliable sensors can save time during debugging by helping you identify errors in your code rather than in the hardware. The project also involves the use of several libraries on the Raspberry Pi, including Adafruit_DHT, Adafruit_Python_DHT, Adafruit_CircuitPython, and Adafruit_Python_BMP. These libraries are essential for retrieving data from the DHT22 and BMP180 sensors.

By following this guide, you will understand how to effectively build and manage an IoT weather monitoring system, taking into consideration the cost and reliability of sensors, as well as the interaction of various software libraries.

Jun Wang

Arizona State University

Introduction

IoT weather monitoring systems leverage cutting-edge IT technologies, including advanced sensors, portable mini-computers, AWS Cloud services, high-level programming languages, databases, and even AI, to create versatile systems applicable in various scenarios. These systems can record data for climate change studies and support future research through AI models like ANN (Artificial Neural Networks) or the Random Forest method, aiding in weather prediction and natural disaster prevention.

For instance, an IoT weather monitoring system could be deployed in California to help prevent forest fires. Historical data shows that the top seven worst wildfires in California each resulted in several billion dollars in insured losses. The 2018 Camp Fire alone caused estimated damages of $10 billion, or approximately $10.38 billion in 2020 value. Deploying an IoT weather monitoring system in forests could enable early detection of high temperatures through sensors, which would then alert fire stations or weather monitoring stations. These agencies could respond promptly to these alerts, potentially preventing fires in their early stages.

This paper will guide you through the process of building an IoT weather monitoring system. You will learn how to select appropriate sensors, connect them to a Raspberry Pi, test their functionality, and use Python code to read data from these sensors. Additionally, you will learn how to transmit data to a server via MQTT and troubleshoot any errors that arise. By the end of this paper, you will have a comprehensive understanding of IoT systems. The knowledge and experience gained from this project can be applied to other IoT applications, such as smart lighting systems or air pollution monitoring systems.

Design Process

- Software and Hardware Preparation

Before we enter design process, the software and hardware requirement need to show first.

Software requirement	Hardware requirement
Python Charm IDE	Raspberry Pi 4
MySQL database	DHT22 Sensor
Restful - fastAPI	BMP180 Sensor
Python3	Breadboard
BMP180 and DHT22 library(adafruit)	Jumper wires
Window/Linux/Raspbian	

When building a weather IoT system, cost is a crucial factor. Different sensors come at varying prices, and using unreliable sensors can lead to frequent replacements, increasing both time and budget costs. Poor-quality sensors can also send inaccurate data to servers, affecting weather information accuracy and future research outcomes.

For purchasing reliable sensors and devices, Amazon is an excellent choice. You can read reviews from other users and compare prices to make informed decisions. For instance, HiLetgo offers high-quality DHT22 and BMP180 sensors. You can purchase 5 pieces of BMP280 sensors for $6 and 2 pieces of DHT22 sensors for $13. Additionally, AITRIP provides

10 pieces of BMP180 sensors for $8. All of these options are reliable.

For a powerful and portable mini-computer, the CanaKit Raspberry Pi 4 Extreme Kit is recommended. Currently, it is challenging to find Raspberry Pi 3 on Amazon. If you need to buy a Raspberry Pi 3, you might have to visit the CanaKit official website, where delivery may take a few weeks. To ensure timely project progress, Amazon's accurate and fast delivery can be advantageous.

- IoT Level

In this project, the IoT level 3 is chosen for building the weather monitor system. A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and the application is cloud-based. Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive. (Bahga, 2014) Based on these characteristics, IoT level 3 is suitable for this IoT weather monitor system.

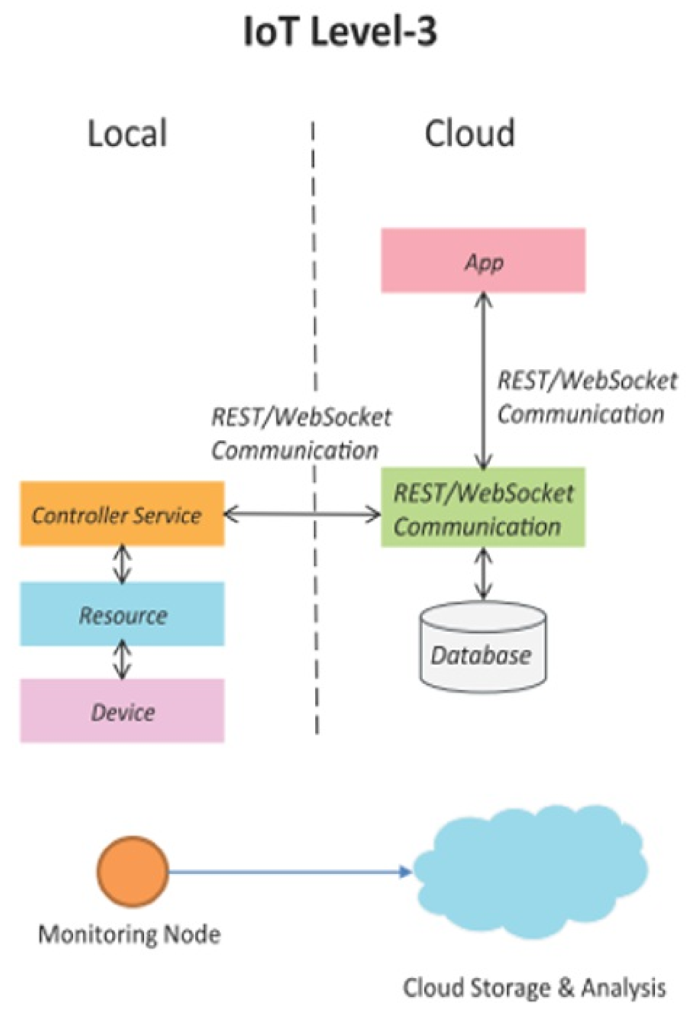


Figure 1. (Bahga, 2014)

- Block diagram

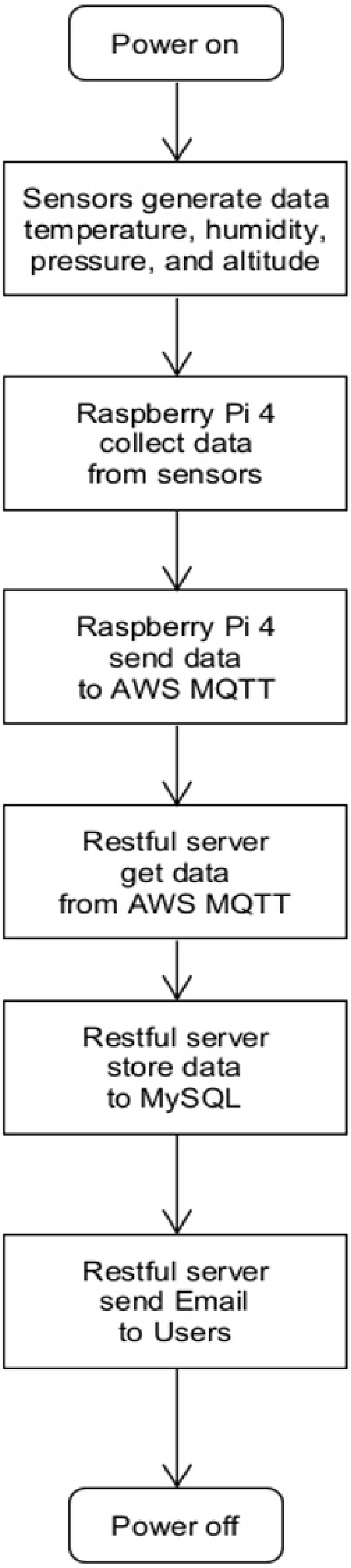

Power on
Sensors generate data temperature, humidity, pressure, and altitude
Raspberry Pi 4 collect data from sensors
Raspberry Pi 4 send data to AWS MQTT
Restful server get data from AWS MQTT
Restful server store data to MySQL
Restful server send Email to Users
Power off

Figure 2.

- Data flow charts

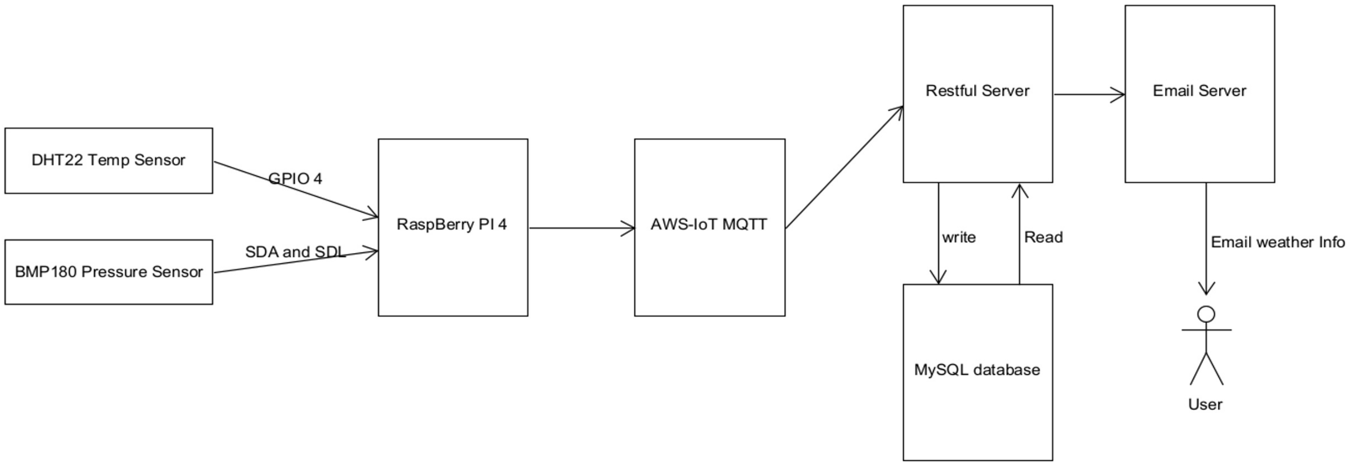


Figure 3.

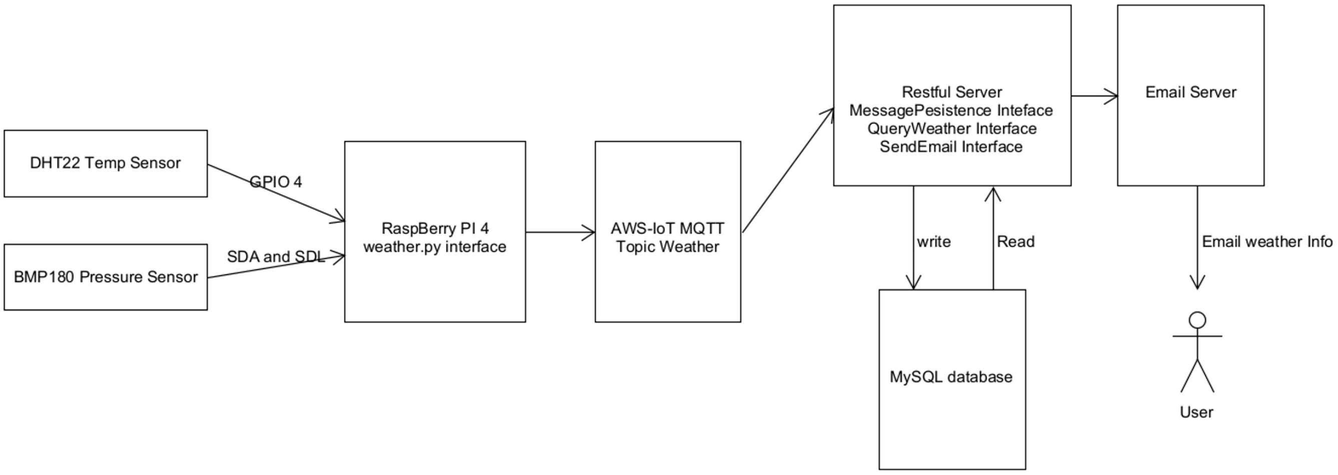
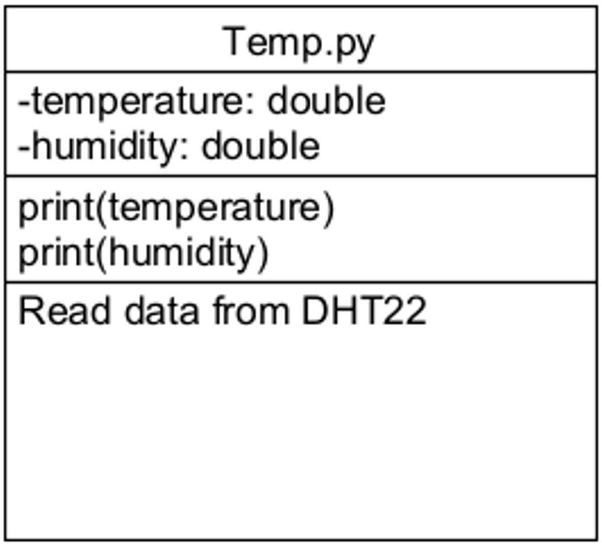


Figure 4.

- Interface design

Raspberry Pi 4 side:

Temp.py



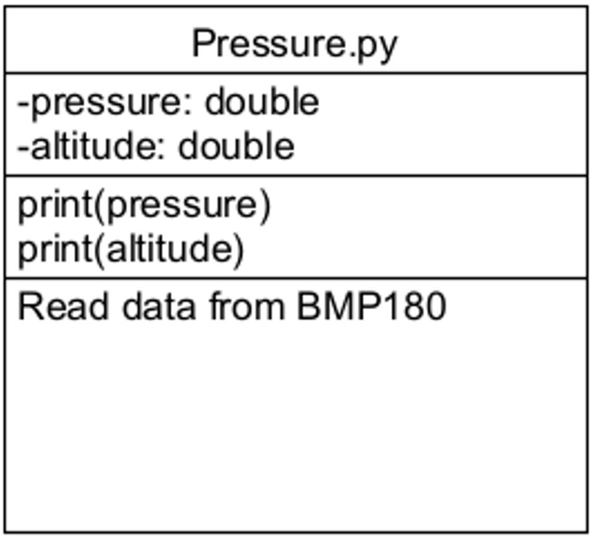
Temp.py

-temperature: double
-humidity: double

print(temperature)
print(humidity)

Read data from DHT22

Pressure.py



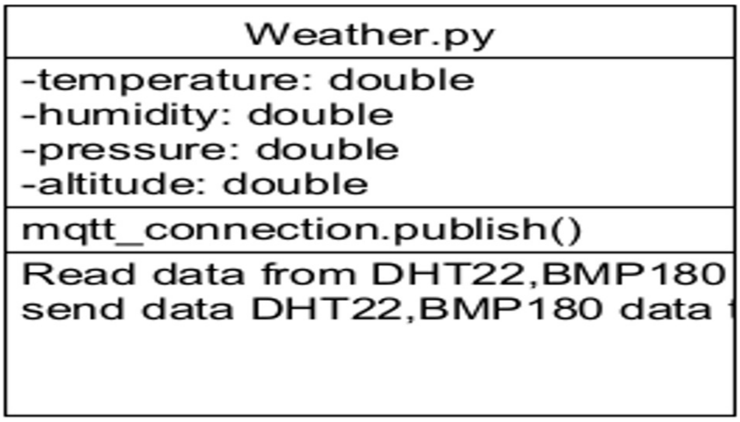
Pressure.py

-pressure: double
-altitude: double

print(pressure)
print(altitude)

Read data from BMP180

Weather.py



Weather.py

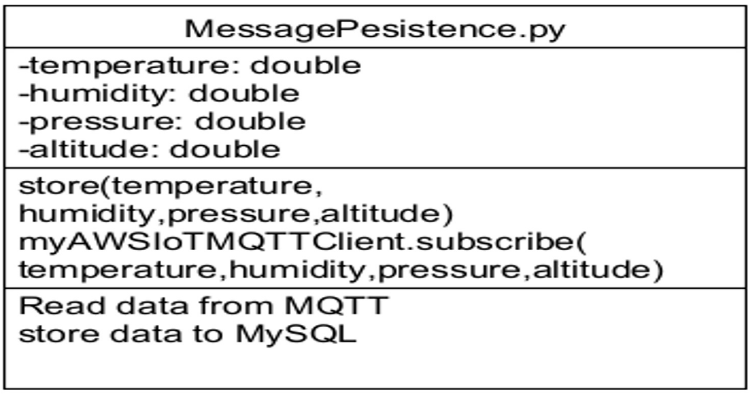
-temperature: double
-humidity: double
-pressure: double
-altitude: double

mqtt_connection.publish()

Read data from DHT22,BMP180
send data DHT22,BMP180 data t

Restful servers' side:

MessagePesistence.py



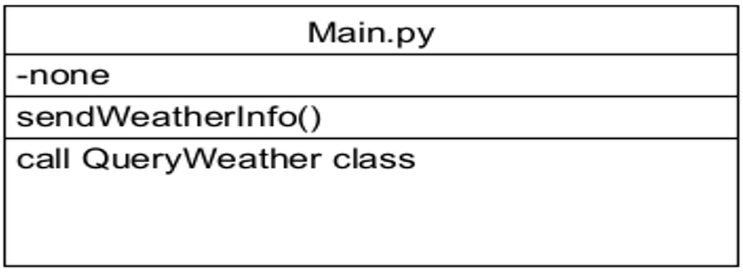
MessagePesistence.py

-temperature: double
-humidity: double
-pressure: double
-altitude: double

store(temperature,
humidity,pressure,altitude)
myAWSIoTMQTTClient.subscribe(
temperature,humidity,pressure,altitude)

Read data from MQTT
store data to MySQL

Main.py



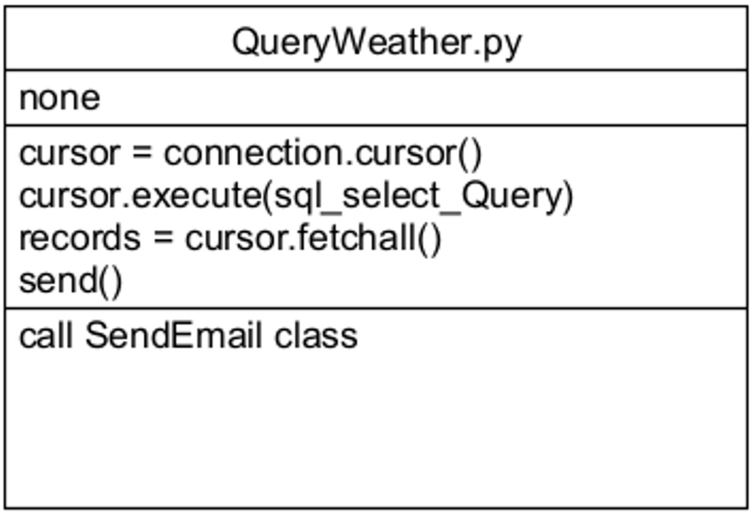
Main.py

-none

sendWeatherInfo()

call QueryWeather class

QueryWeather.py



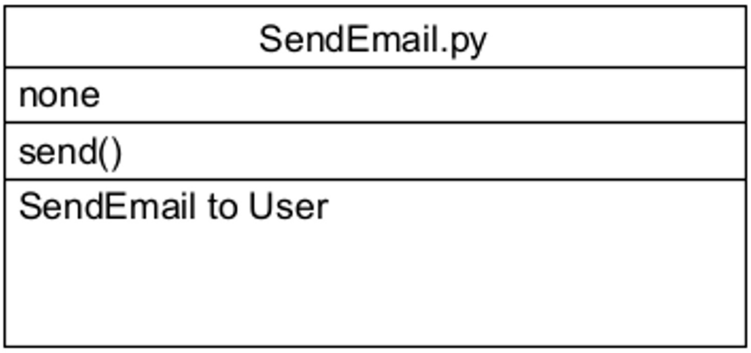
QueryWeather.py

none

cursor = connection.cursor()
cursor.execute(sql_select_Query)
records = cursor.fetchall()
send()

call SendEmail class

SendEmail.py



SendEmail.py

none

send()

SendEmail to User

Development Process

In the figure below, you can see DHT22 installed on the breadboard. The DHT22 is a powerful sensor to record temperature, humility. Some jumper wires are used to connect Raspberry PI 4.

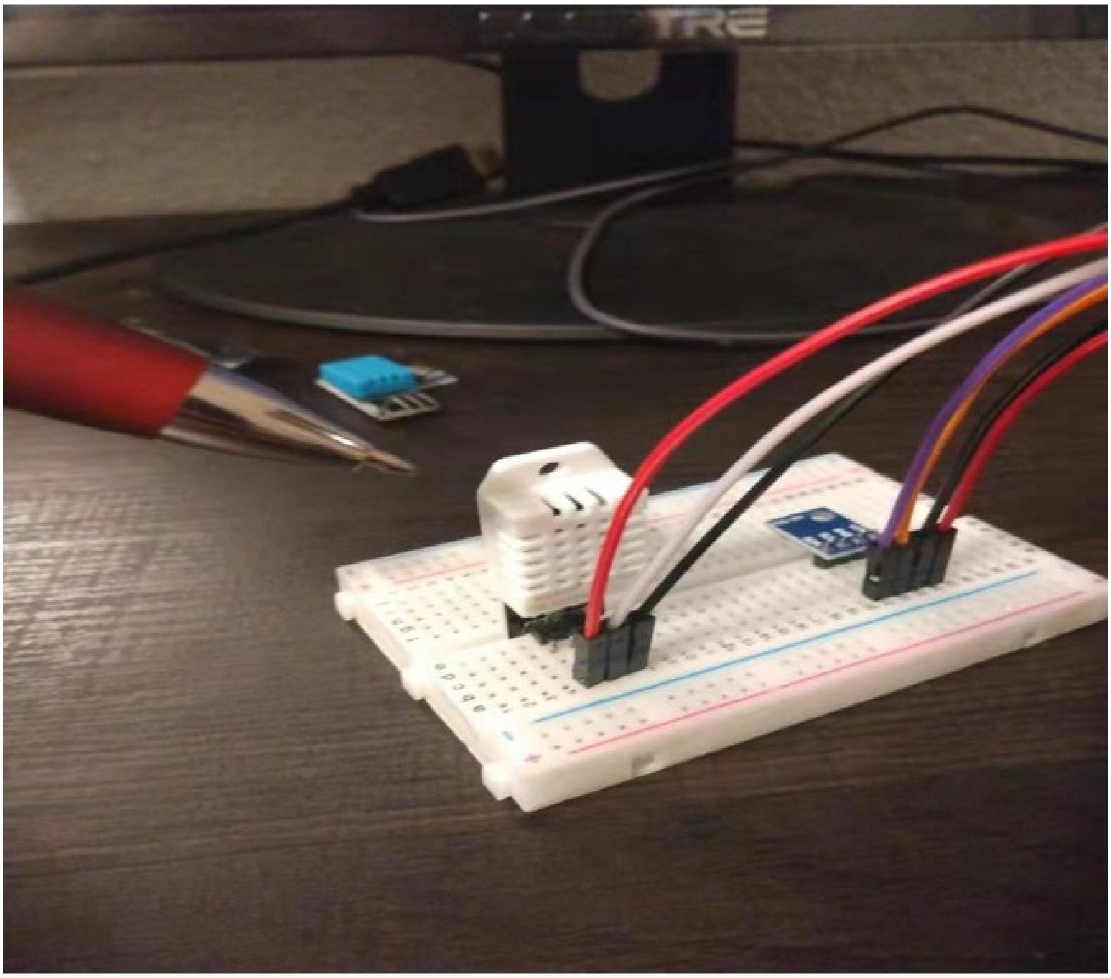


Figure 5.

For connection to Raspberry PI 4, as the below figure 6, the positive port in the DHT22 must connect to the 5V power port. Negative port must connect to the ground port through jumper wires. Here, data out ports can connect to GPIO4.

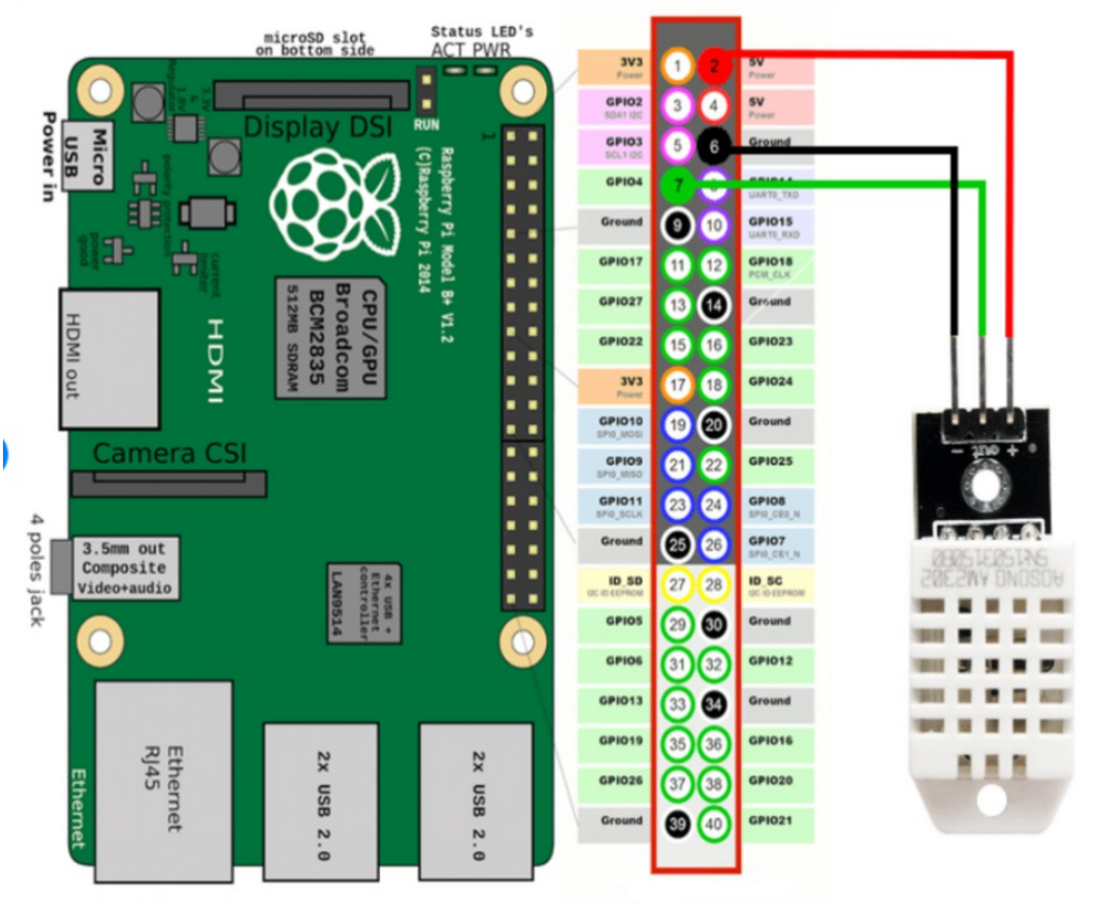

microSD slot on bottom side
Status LED's ACT PWR
Power in
Micro USB
Display DSI
RUN
Raspberry Pi Model B+ V1.2
(C)Raspberry Pi 2014
CPU/GPU Broadcom BCM2835 512MB SDRAM
HDMI
HDMI out
Camera CSI
4 poles jack
3.5mm out Composite Video+audio
4x USB + Ethernet controller LAN9514
Ethernet
Ethernet RJ45
2x USB 2.0
2x USB 2.0
3V3 Power 1 2 5V Power
GPIO2 SDA1 I2C 3 4 5V Power
GPIO3 SCL1 I2C 5 6 Ground
GPIO4 7 8 GPIO14 UART0_TXD
Ground 9 10 GPIO15 UART0_RXD
GPIO17 11 12 GPIO18 PCM_CLK
GPIO27 13 14 Ground
GPIO22 15 16 GPIO23
3V3 Power 17 18 GPIO24
GPIO10 SPI0_MOSI 19 20 Ground
GPIO9 SPI0_MISO 21 22 GPIO25
GPIO11 SPI0_SCLK 23 24 GPIO8 SPI0_CE0_N
Ground 25 26 GPIO7 SPI0_CE1_N
ID_SD I2C ID EEPROM 27 28 ID_SC I2C ID EEPROM
GPIO5 29 30 Ground
GPIO6 31 32 GPIO12
GPIO13 33 34 Ground
GPIO19 35 36 GPIO16
GPIO26 37 38 GPIO20
Ground 39 40 GPIO21
- out +
AOSONG AM2302

Figure 6.

To test whether the DHT22 is working or not, first, you need to install Adafruit_DHT. And then, you need to add below code to Adafruit_DHT/platform_detect.py

elif match.group(1) == 'BCM2711':

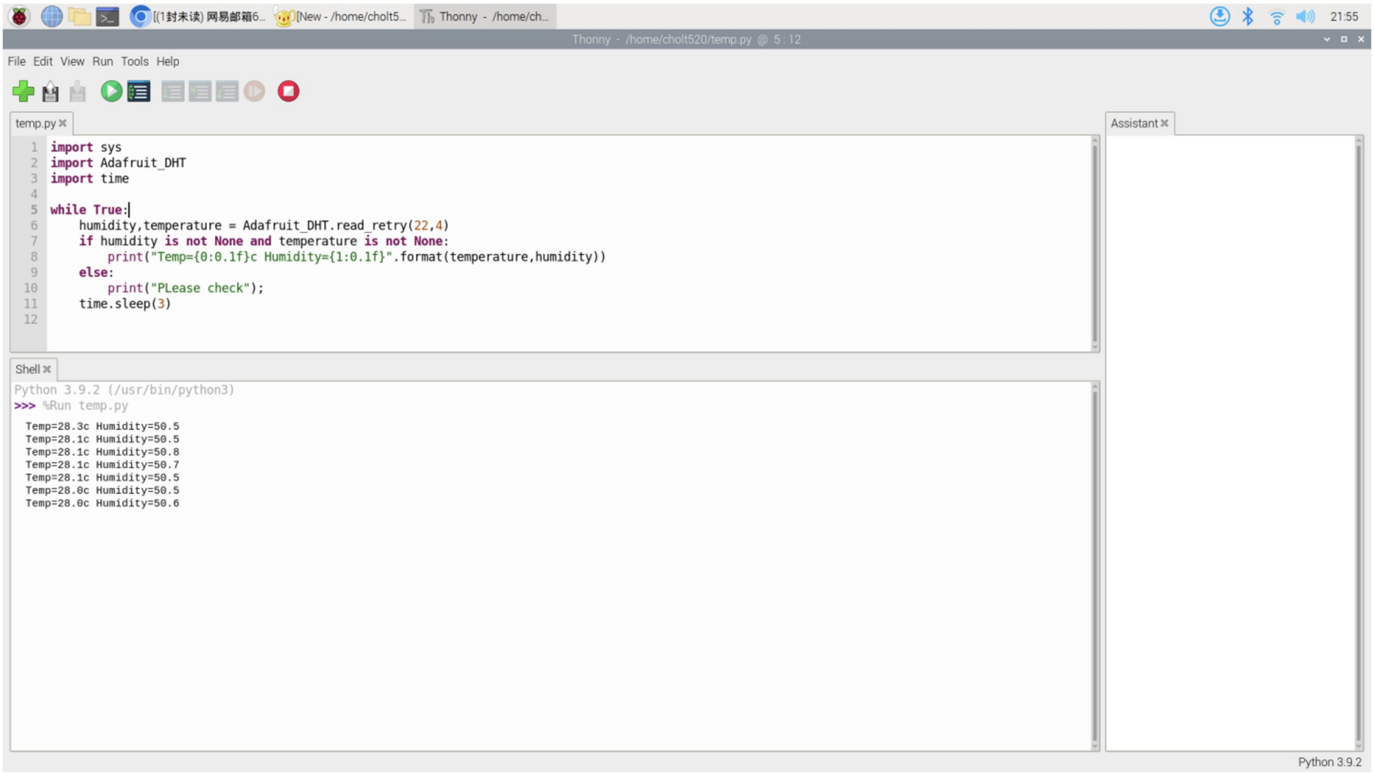
#Pi 4b

return 3

You can write the code as below figure 7. The code means read data from DHT22 and GPIO4 return in variable humidity and temperature.

```
humidity,temperature = Adafruit_DHT.read_retry(22,4)
```

humidity, temperature = Adafruit_DHT.read_retry(22,4)



```python
import sys
import Adafruit_DHT
import time

while True:
    humidity,temperature = Adafruit_DHT.read_retry(22,4)
    if humidity is not None and temperature is not None:
        print("Temp={0:0.1f}c Humidity={1:0.1f}".format(temperature,humidity))
    else:
        print("PLease check");
    time.sleep(3)
```

```
Python 3.9.2 (/usr/bin/python3)
>>> %Run temp.py

  Temp=28.3c Humidity=50.5
  Temp=28.1c Humidity=50.5
  Temp=28.1c Humidity=50.8
  Temp=28.1c Humidity=50.7
  Temp=28.1c Humidity=50.5
  Temp=28.0c Humidity=50.5
  Temp=28.0c Humidity=50.6
```

Figure 7.

In the below figure 8, you can see BMP180 installed on the breadboard. The BMP180 is a powerful sensor to record pressure, altitude, and temp. Some jumper wires are used to connect Raspberry PI 4.

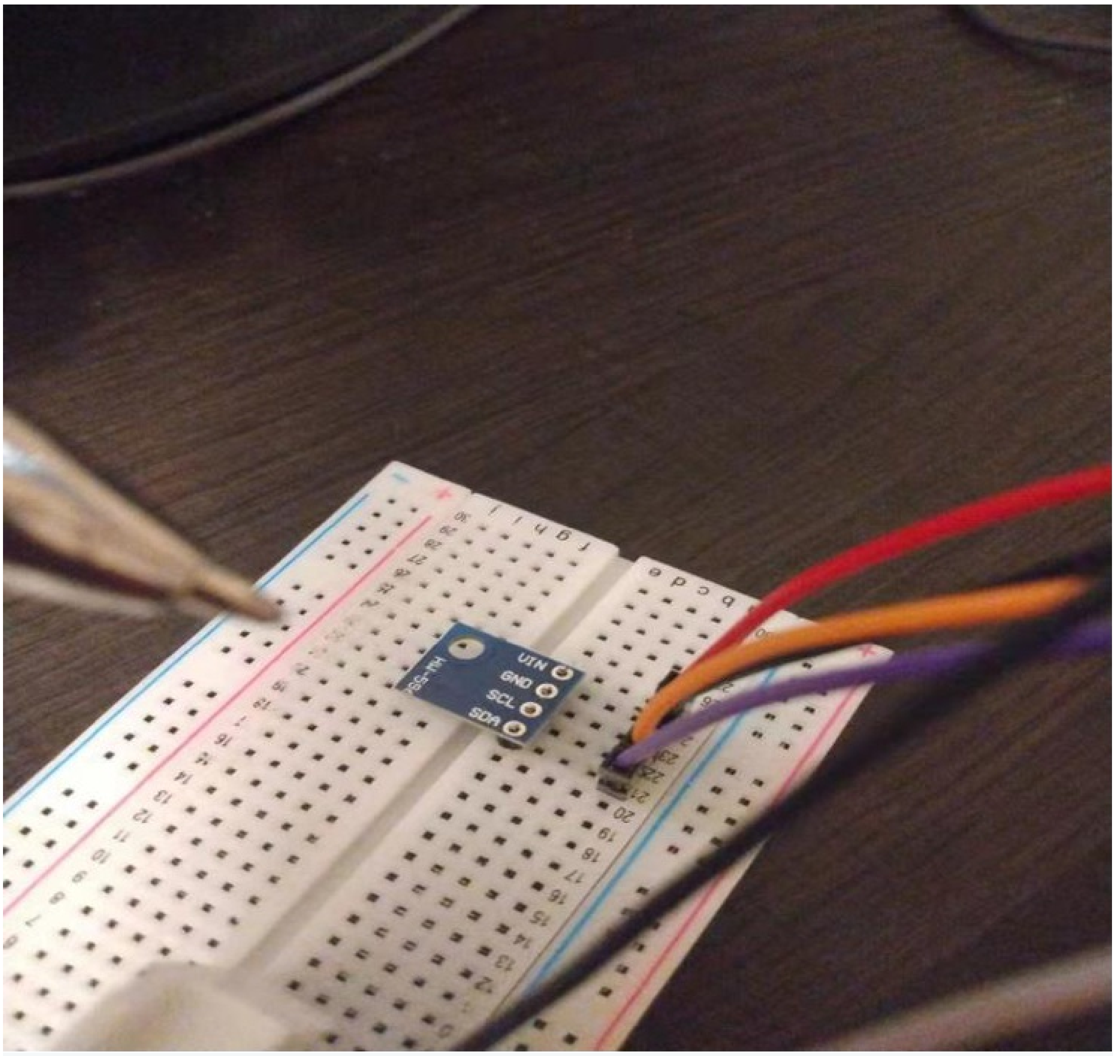


Figure 8.

For connection to Raspberry PI 4, as the below figure 9, the positive port in the BMP180 must connect to the 3V3 power port. Negative port must connect to the ground port through jumper wires. Here, data out ports can connect to SCL and SDA ports.

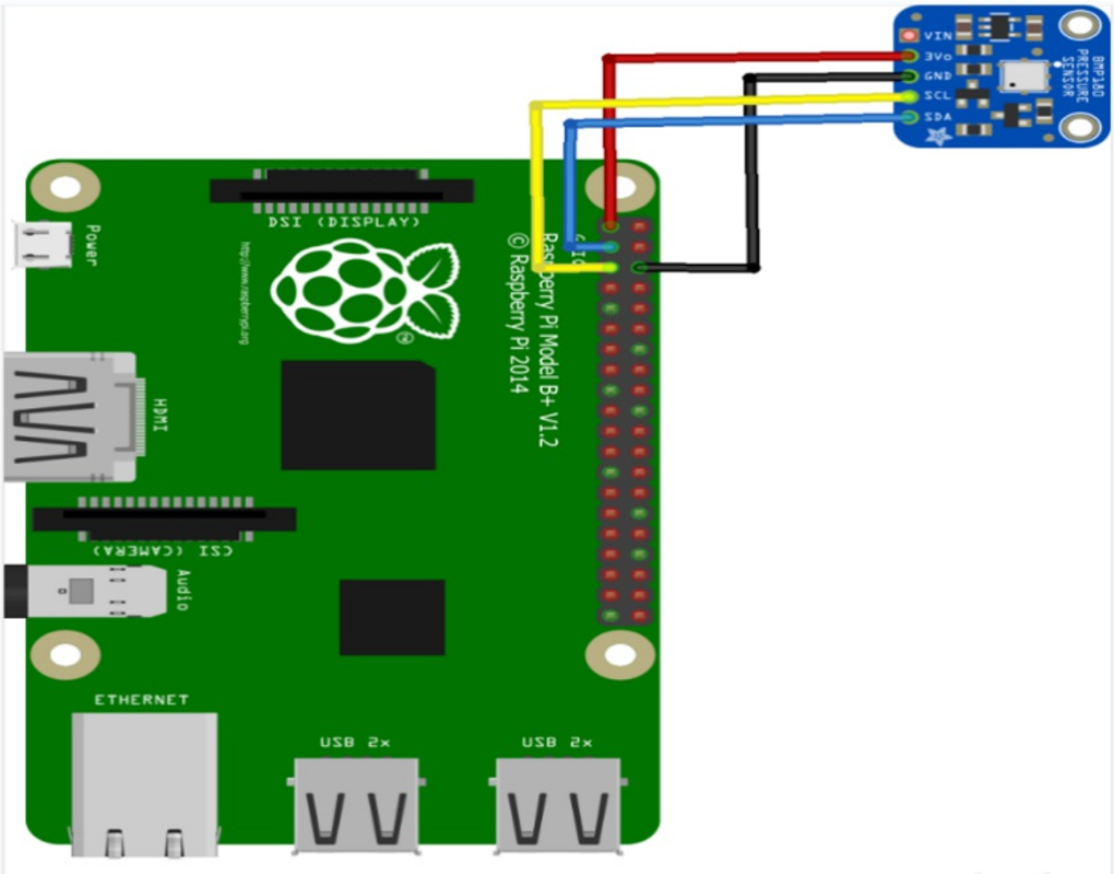
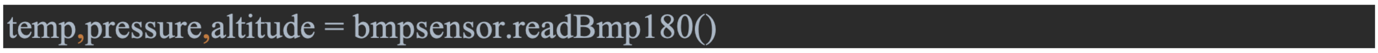


Figure 9.

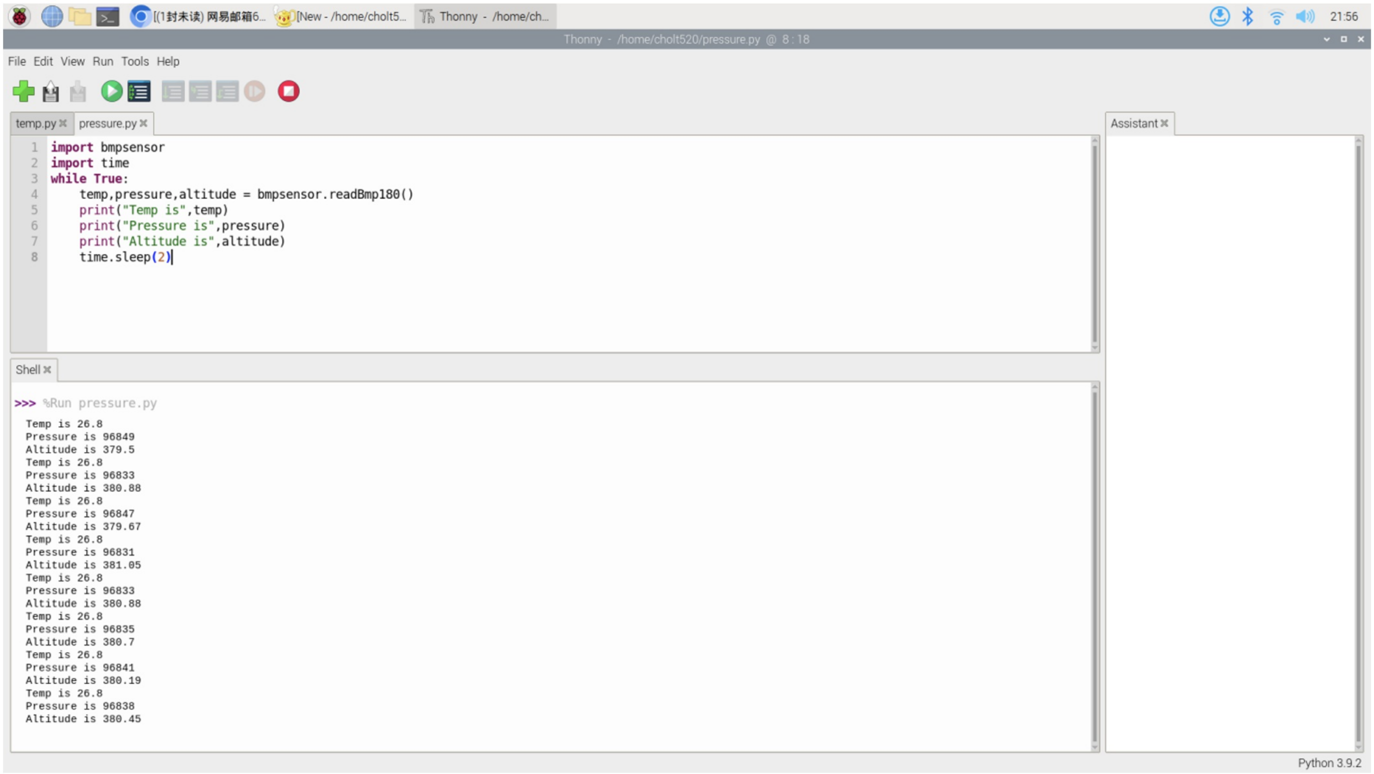
To test whether the BMP180 is working or not, first, you need to install Adafruit_BMP.

You can write the code as below figure 10. The code means to read data from BMP180 return in variable pressure and altitude.



```
temp,pressure,altitude = bmpsensor.readBmp180()
```

temp, pressure, altitude = bmpsensor.readBmp180()



```
import bmpsensor
import time
while True:
    temp,pressure,altitude = bmpsensor.readBmp180()
    print("Temp is",temp)
    print("Pressure is",pressure)
    print("Altitude is",altitude)
    time.sleep(2)
```

```
>>> %Run pressure.py
  Temp is 26.8
  Pressure is 96849
  Altitude is 379.5
  Temp is 26.8
  Pressure is 96833
  Altitude is 380.88
  Temp is 26.8
  Pressure is 96847
  Altitude is 379.67
  Temp is 26.8
  Pressure is 96831
  Altitude is 381.05
  Temp is 26.8
  Pressure is 96833
  Altitude is 380.88
  Temp is 26.8
  Pressure is 96835
  Altitude is 380.7
  Temp is 26.8
  Pressure is 96841
  Altitude is 380.19
  Temp is 26.8
  Pressure is 96838
  Altitude is 380.45
```

Figure 10.

When you finish the previous work, you will get the IoT hardware like below figure 11.

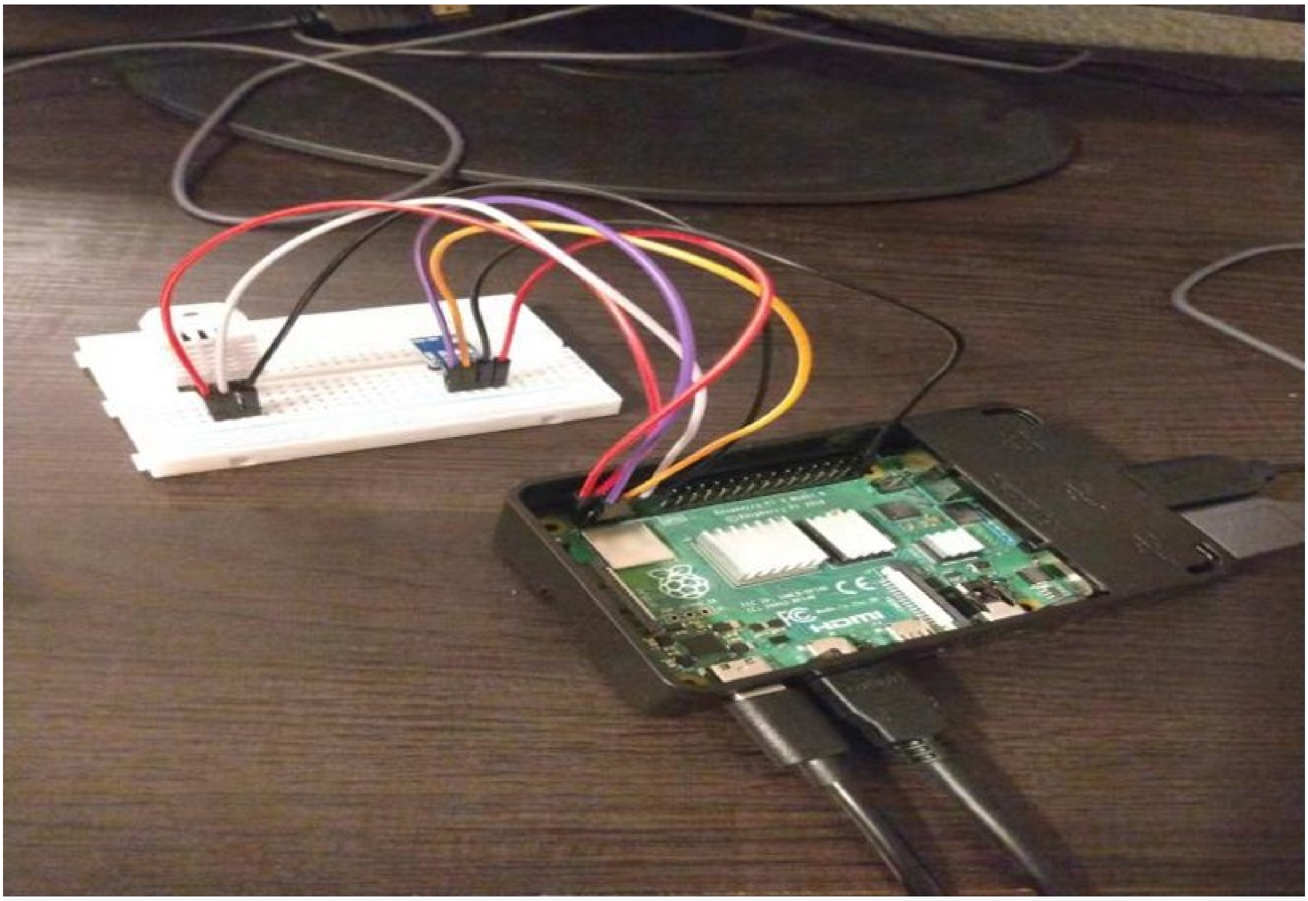
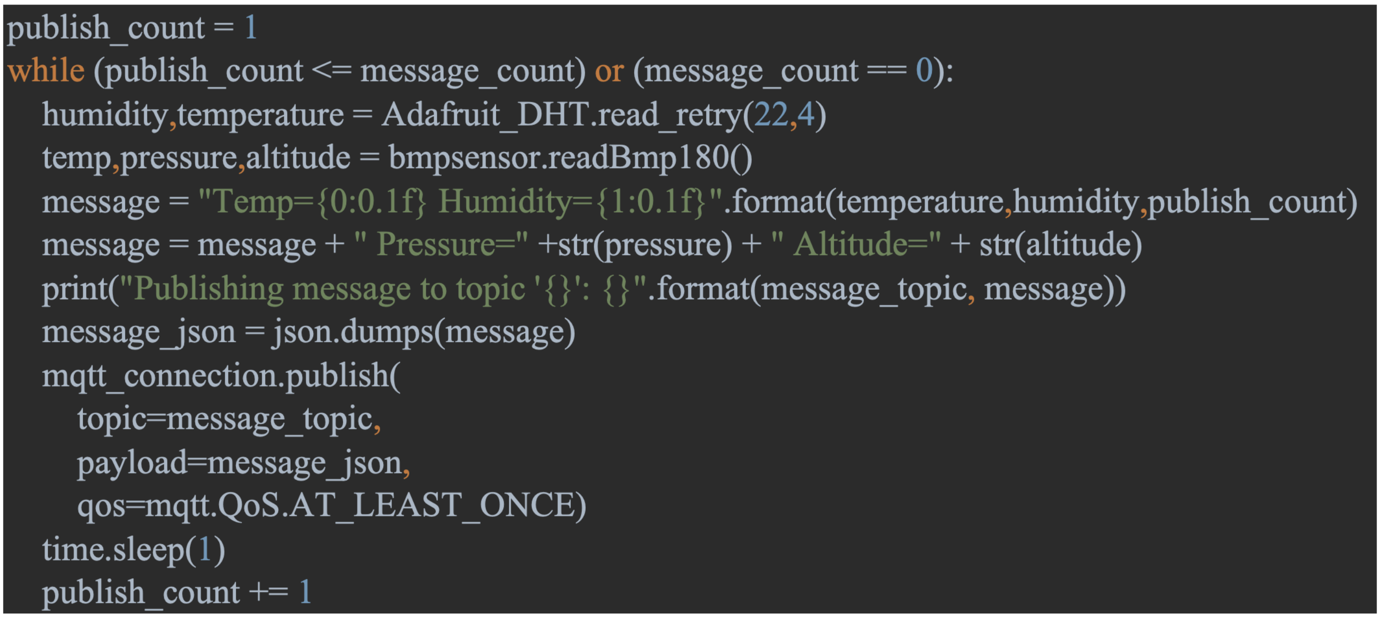


Figure 11.

Here, the Raspberry PI 4 needs to send data to AWS-IOT-MQTT. First, you need to install awsiotsdk. The below code will send data from sensors to MQTT through mqtt_connect.publish() function.



```
publish_count = 1
while (publish_count <= message_count) or (message_count == 0):
    humidity,temperature = Adafruit_DHT.read_retry(22,4)
    temp,pressure,altitude = bmpsensor.readBmp180()
    message = "Temp={0:0.1f} Humidity={1:0.1f}".format(temperature,humidity,publish_count)
    message = message + " Pressure=" +str(pressure) + " Altitude=" + str(altitude)
    print("Publishing message to topic '{}': {}".format(message_topic, message))
    message_json = json.dumps(message)
    mqtt_connection.publish(
        topic=message_topic,
        payload=message_json,
        qos=mqtt.QoS.AT_LEAST_ONCE)
    time.sleep(1)
    publish_count += 1
```

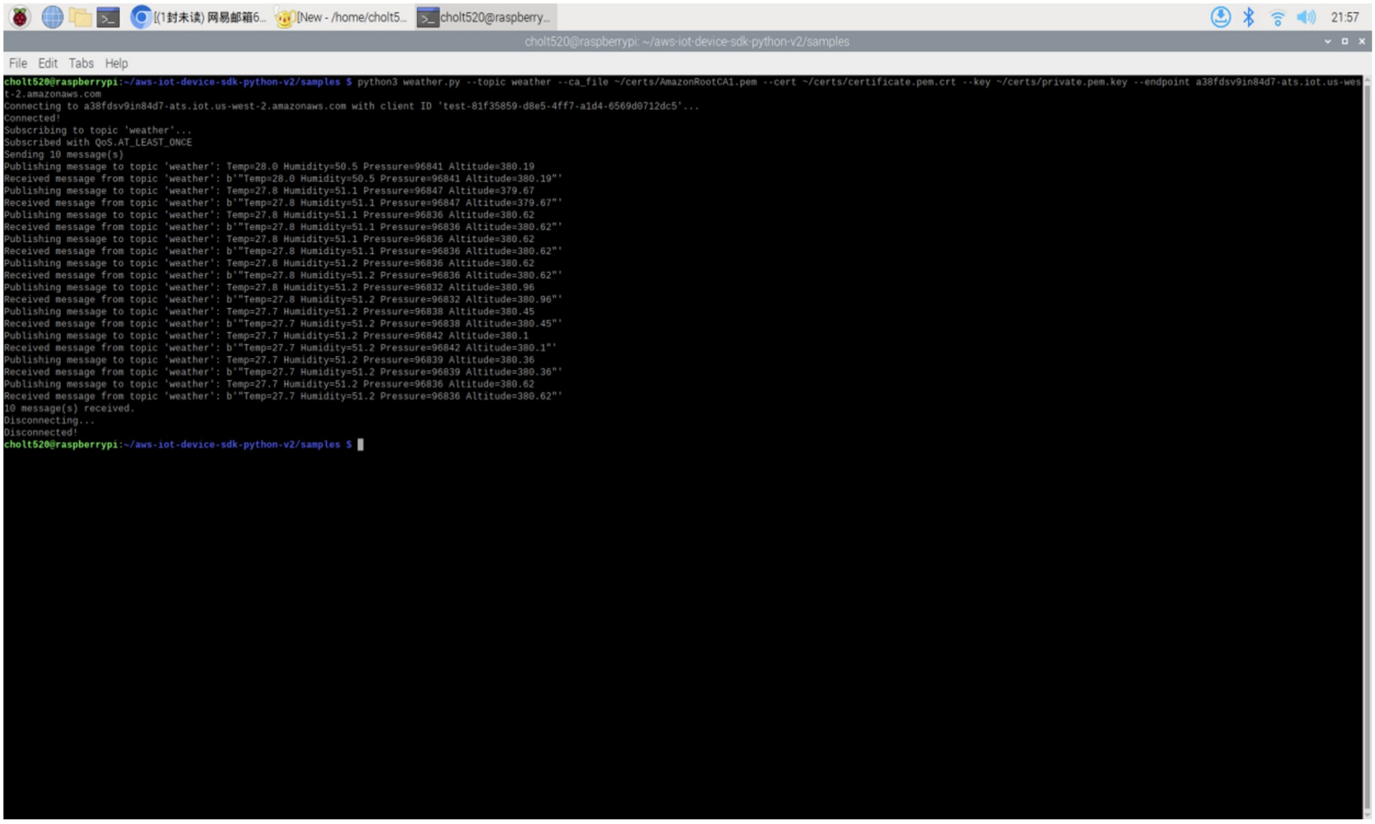
```
publish_count = 1
while (publish_count <= message_count) or (message_count == 0):
```

```
humidity, temperature = Adafruit_DHT.read_retry(22,4)
temp, pressure, altitude = bmpsensor.readBmp180()
message = "Temp={0:0.1f} Humidity={1:0.1f}".format(temperature, humidity, publish_count)
message = message + " Pressure=" +str(pressure) + " Altitude=" + str(altitude)
print("Publishing message to topic '{}': {}".format(message_topic, message))
message_json = json.dumps(message)
mqtt_connection.publish(
    topic=message_topic,
    payload=message_json,
    qos=mqtt.QoS.AT_LEAST_ONCE)
time.sleep(1)
publish_count += 1
```

When you finish the code, you can run command:

python3 weather.py --topic weather --ca_file ~/certs/AmazonRootCA1.pem --cert ~/certs/certificate.pem.crt --key ~/certs/private.pem.key --endpoint a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com

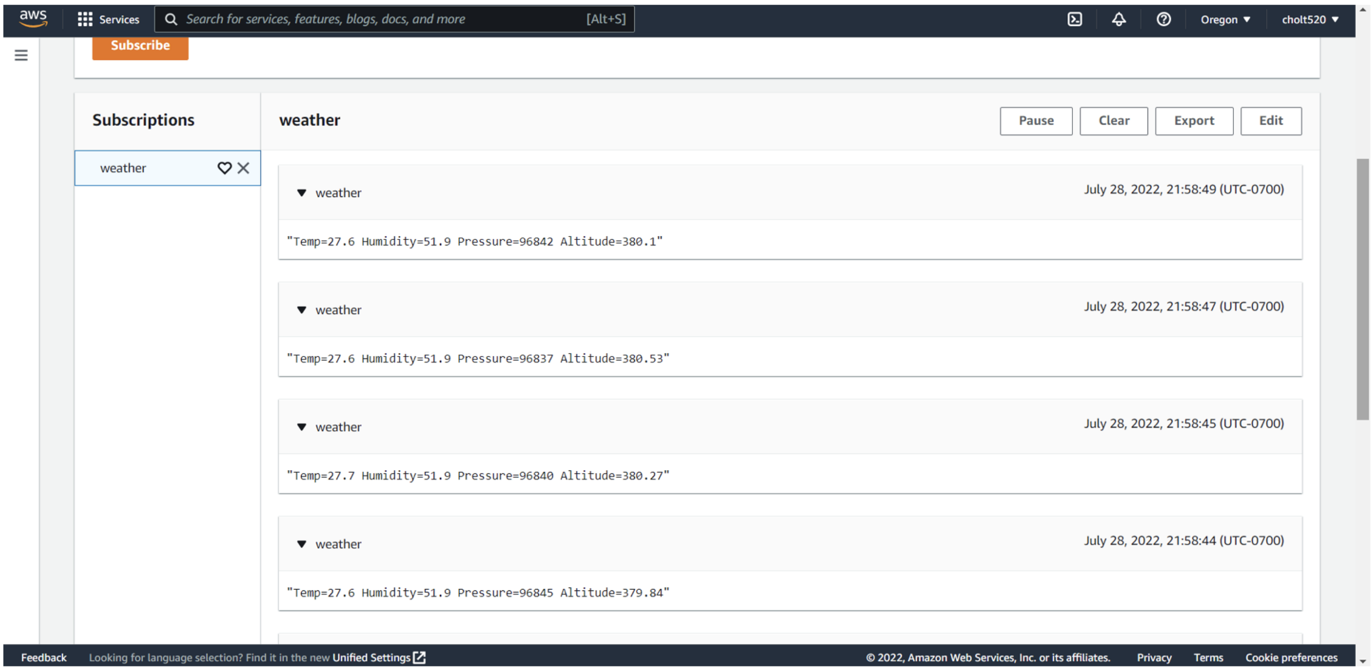
Now, you can see the data is sending to AWS-MQTT



```
cholt520@raspberrypi:~/aws-iot-device-sdk-python-v2/samples $ python3 weather.py --topic weather --ca_file ~/certs/AmazonRootCA1.pem --cert ~/certs/certificate.pem.crt --key ~/certs/private.pem.key --endpoint a38fdsv9in84d7-ats.iot.us-wes
t-2.amazonaws.com
Connecting to a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com with client ID 'test-81f35859-d8e5-4ff7-a1d4-6569d0712dc5'...
Connected!
Subscribing to topic 'weather'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'weather': Temp=28.0 Humidity=50.5 Pressure=96841 Altitude=380.19
Received message from topic 'weather': b'"Temp=28.0 Humidity=50.5 Pressure=96841 Altitude=380.19"'
Publishing message to topic 'weather': Temp=27.8 Humidity=51.1 Pressure=96847 Altitude=379.67
Received message from topic 'weather': b'"Temp=27.8 Humidity=51.1 Pressure=96847 Altitude=379.67"'
Publishing message to topic 'weather': Temp=27.8 Humidity=51.1 Pressure=96836 Altitude=380.62
Received message from topic 'weather': b'"Temp=27.8 Humidity=51.1 Pressure=96836 Altitude=380.62"'
Publishing message to topic 'weather': Temp=27.8 Humidity=51.1 Pressure=96836 Altitude=380.62
Received message from topic 'weather': b'"Temp=27.8 Humidity=51.1 Pressure=96836 Altitude=380.62"'
Publishing message to topic 'weather': Temp=27.8 Humidity=51.2 Pressure=96836 Altitude=380.62
Received message from topic 'weather': b'"Temp=27.8 Humidity=51.2 Pressure=96836 Altitude=380.62"'
Publishing message to topic 'weather': Temp=27.8 Humidity=51.2 Pressure=96832 Altitude=380.96
Received message from topic 'weather': b'"Temp=27.8 Humidity=51.2 Pressure=96832 Altitude=380.96"'
Publishing message to topic 'weather': Temp=27.7 Humidity=51.2 Pressure=96838 Altitude=380.45
Received message from topic 'weather': b'"Temp=27.7 Humidity=51.2 Pressure=96838 Altitude=380.45"'
Publishing message to topic 'weather': Temp=27.7 Humidity=51.2 Pressure=96842 Altitude=380.1
Received message from topic 'weather': b'"Temp=27.7 Humidity=51.2 Pressure=96842 Altitude=380.1"'
Publishing message to topic 'weather': Temp=27.7 Humidity=51.2 Pressure=96839 Altitude=380.36
Received message from topic 'weather': b'"Temp=27.7 Humidity=51.2 Pressure=96839 Altitude=380.36"'
Publishing message to topic 'weather': Temp=27.7 Humidity=51.2 Pressure=96836 Altitude=380.62
Received message from topic 'weather': b'"Temp=27.7 Humidity=51.2 Pressure=96836 Altitude=380.62"'
10 message(s) received.
Disconnecting...
Disconnected!
cholt520@raspberrypi:~/aws-iot-device-sdk-python-v2/samples $ 
```

Figure 12.

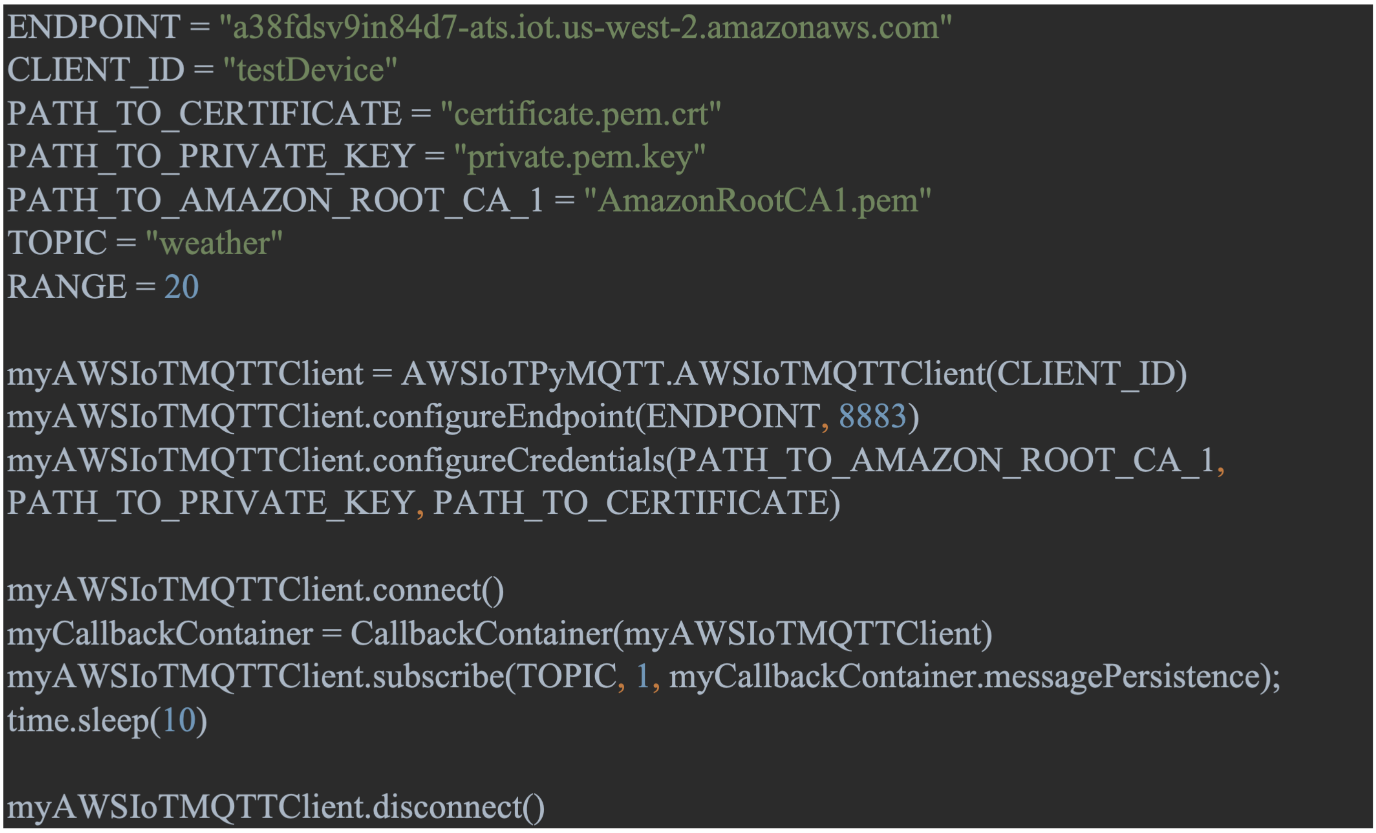
For the AWS-MQTT console, you need to subscribe to the topic: weather. And then you can see the data which comes from IoT devices.


aws
Services
Search for services, features, blogs, docs, and more
[Alt+S]
Oregon
cholt520
Subscribe
Subscriptions
weather
weather
Pause
Clear
Export
Edit
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

```
weather    July 28, 2022, 21:58:49 (UTC-0700)
"Temp=27.6 Humidity=51.9 Pressure=96842 Altitude=380.1"

weather    July 28, 2022, 21:58:47 (UTC-0700)
"Temp=27.6 Humidity=51.9 Pressure=96837 Altitude=380.53"

weather    July 28, 2022, 21:58:45 (UTC-0700)
"Temp=27.7 Humidity=51.9 Pressure=96840 Altitude=380.27"

weather    July 28, 2022, 21:58:44 (UTC-0700)
"Temp=27.6 Humidity=51.9 Pressure=96845 Altitude=379.84"
```

Figure 13.

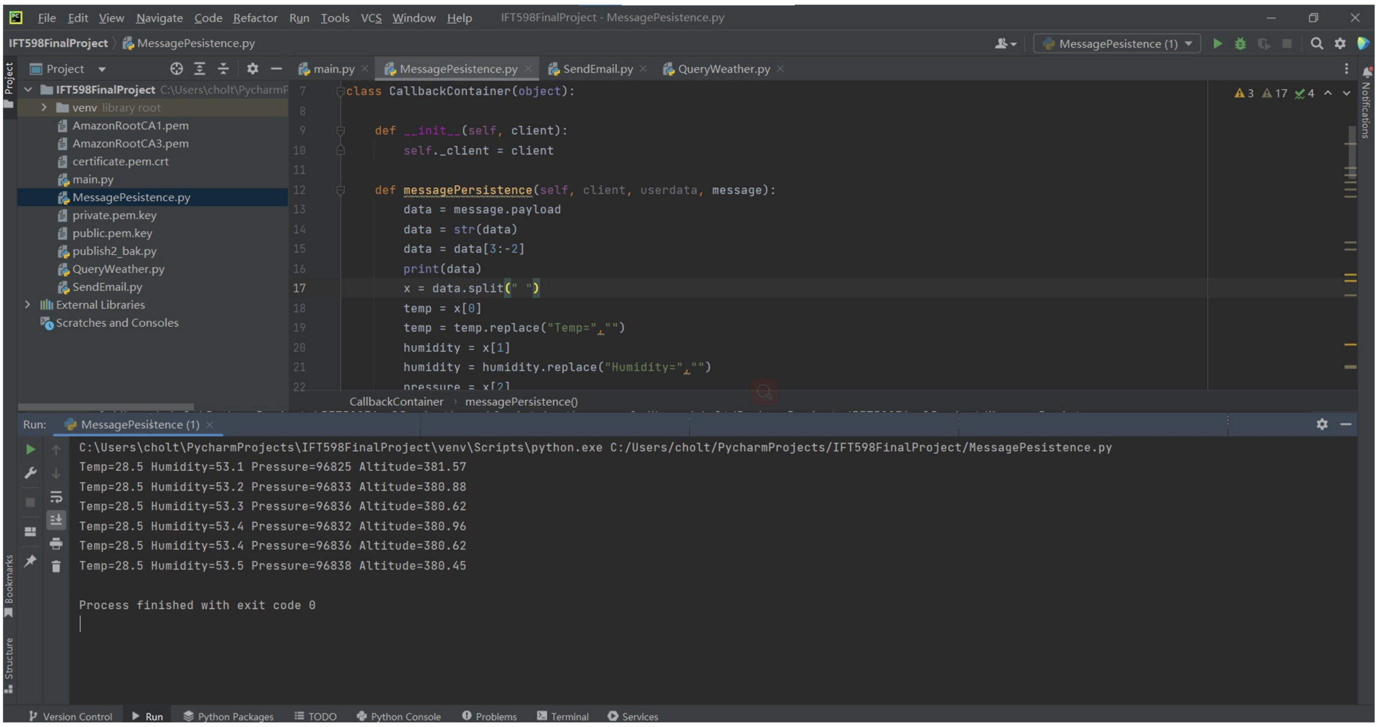
In the below figure 14, it shows the Restful server reading data from MQTT. First, you need to put certificate.pem.crt, private.pem.key and AmazonRootCA1.pem into the same folder with code. And then use myAWSIoTMQTTClient.subscribe() to read data from MQTT.



```
ENDPOINT = "a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com"
CLIENT_ID = "testDevice"
PATH_TO_CERTIFICATE = "certificate.pem.crt"
PATH_TO_PRIVATE_KEY = "private.pem.key"
PATH_TO_AMAZON_ROOT_CA_1 = "AmazonRootCA1.pem"
TOPIC = "weather"
RANGE = 20

myAWSIoTMQTTClient = AWSIoTPyMQTT.AWSIoTMQTTClient(CLIENT_ID)
myAWSIoTMQTTClient.configureEndpoint(ENDPOINT, 8883)
myAWSIoTMQTTClient.configureCredentials(PATH_TO_AMAZON_ROOT_CA_1,
PATH_TO_PRIVATE_KEY, PATH_TO_CERTIFICATE)

myAWSIoTMQTTClient.connect()
myCallbackContainer = CallbackContainer(myAWSIoTMQTTClient)
myAWSIoTMQTTClient.subscribe(TOPIC, 1, myCallbackContainer.messagePersistence);
time.sleep(10)

myAWSIoTMQTTClient.disconnect()
```

```
ENDPOINT = "a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com"
CLIENT_ID = "testDevice"
PATH_TO_CERTIFICATE = "certificate.pem.crt"
PATH_TO_PRIVATE_KEY = "private.pem.key"
PATH_TO_AMAZON_ROOT_CA_1 = "AmazonRootCA1.pem"
TOPIC = "weather"
RANGE = 20

myAWSIoTMQTTClient = AWSIoTPyMQTT.AWSIoTMQTTClient(CLIENT_ID)
myAWSIoTMQTTClient.configureEndpoint(ENDPOINT, 8883)
myAWSIoTMQTTClient.configureCredentials(PATH_TO_AMAZON_ROOT_CA_1, PATH_TO_PRIVATE_KEY, PATH_TO_CERTIFICATE)

myAWSIoTMQTTClient.connect()
myCallbackContainer = CallbackContainer(myAWSIoTMQTTClient)
myAWSIoTMQTTClient.subscribe(TOPIC, 1, myCallbackContainer.messagePersistence);
time.sleep(10)

myAWSIoTMQTTClient.disconnect()
```

When you get data from MQTT successfully, you can print them out in the Pycharm console as below figure 14.



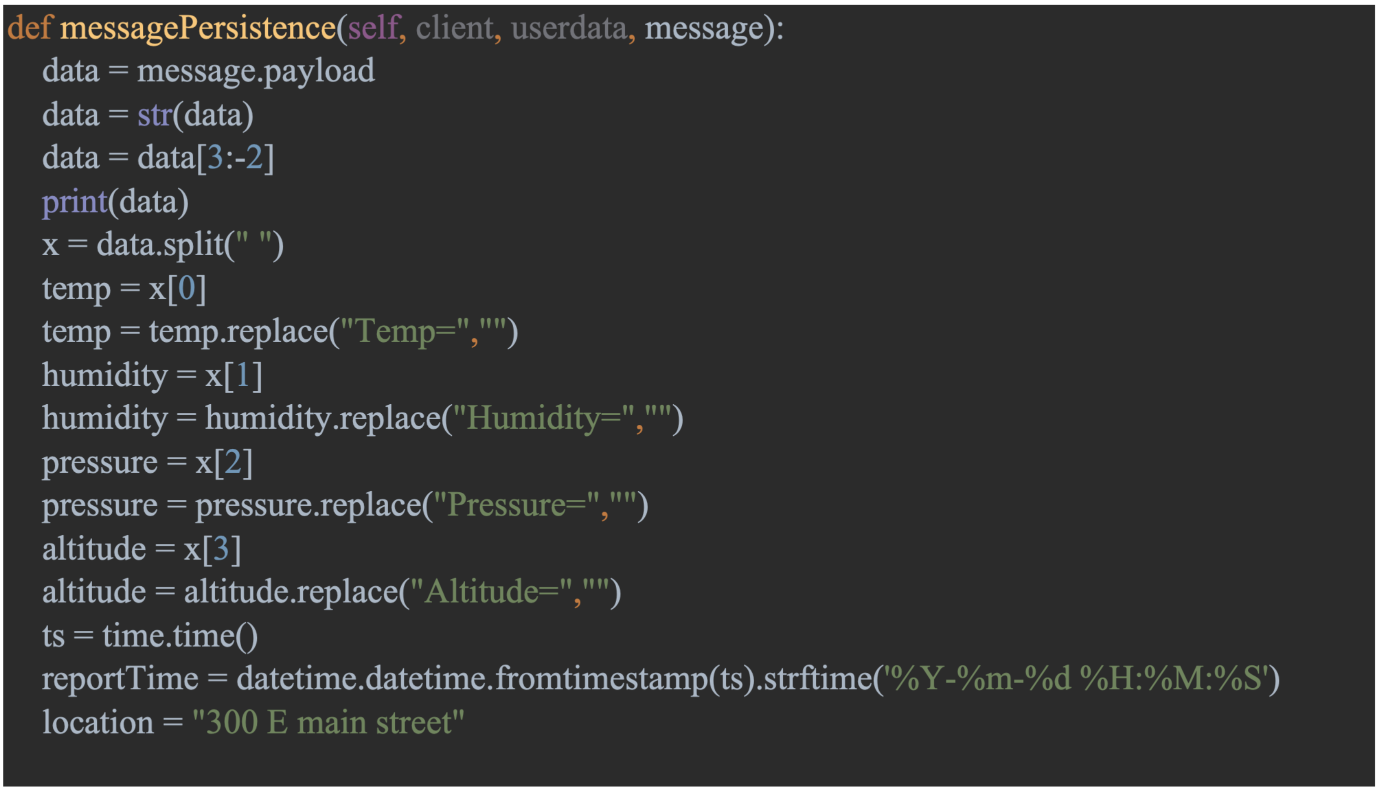
```python
class CallbackContainer(object):

    def __init__(self, client):
        self._client = client

    def messagePersistence(self, client, userdata, message):
        data = message.payload
        data = str(data)
        data = data[3:-2]
        print(data)
        x = data.split(" ")
        temp = x[0]
        temp = temp.replace("Temp=","")
        humidity = x[1]
        humidity = humidity.replace("Humidity=","")
        pressure = x[2]
```

```
C:\Users\cholt\PycharmProjects\IFT598FinalProject\venv\Scripts\python.exe C:/Users/cholt/PycharmProjects/IFT598FinalProject/MessagePesistence.py
Temp=28.5 Humidity=53.1 Pressure=96825 Altitude=381.57
Temp=28.5 Humidity=53.2 Pressure=96833 Altitude=380.88
Temp=28.5 Humidity=53.3 Pressure=96836 Altitude=380.62
Temp=28.5 Humidity=53.4 Pressure=96832 Altitude=380.96
Temp=28.5 Humidity=53.4 Pressure=96836 Altitude=380.62
Temp=28.5 Humidity=53.5 Pressure=96838 Altitude=380.45

Process finished with exit code 0
```

Figure 14.

To store the data into MySQL, you must use the callback function below code.

First, you need to parse the message. This is because the message cannot be directly used without format.

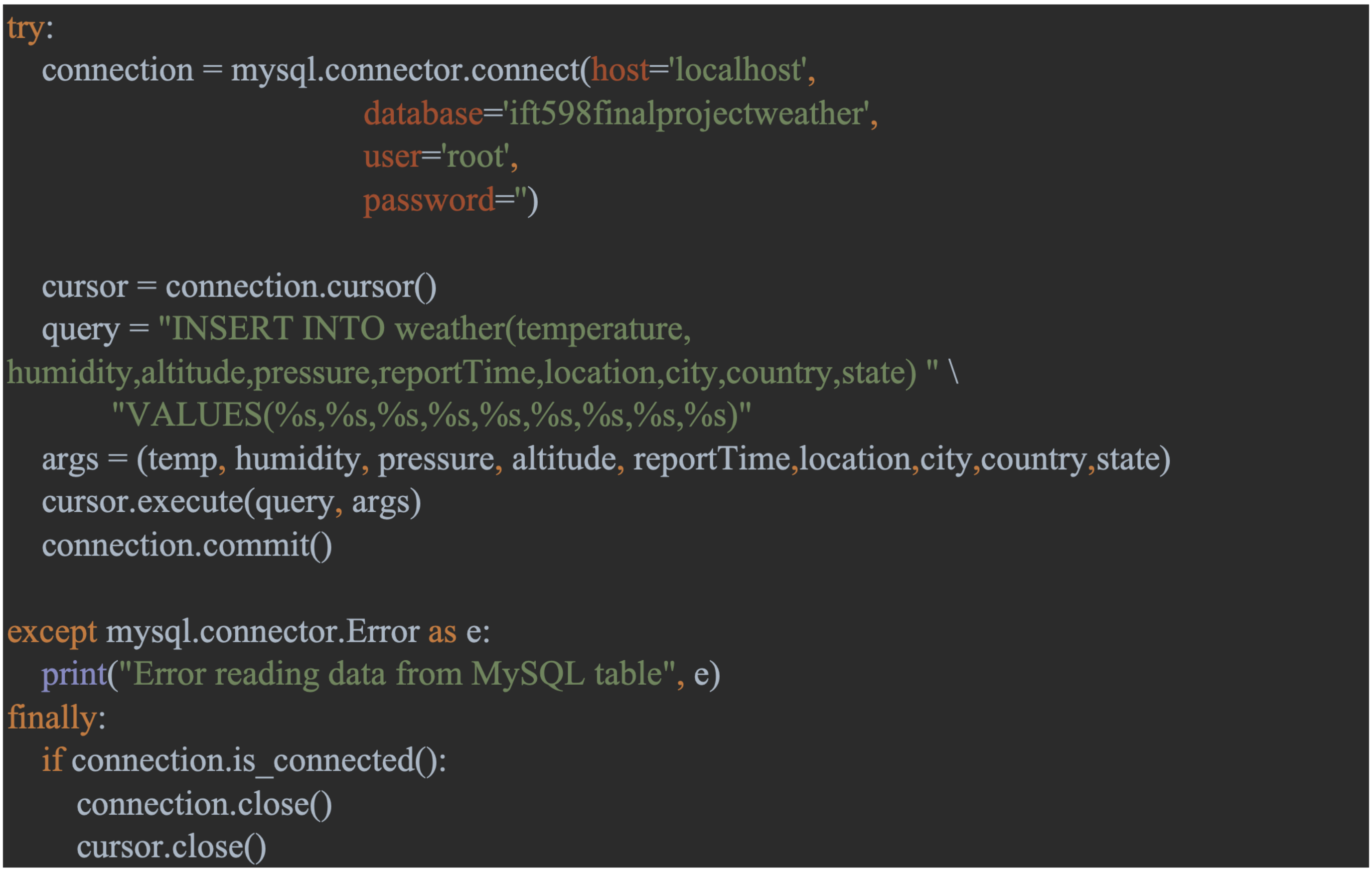


```python
def messagePersistence(self, client, userdata, message):
    data = message.payload
    data = str(data)
    data = data[3:-2]
    print(data)
    x = data.split(" ")
    temp = x[0]
    temp = temp.replace("Temp=","")
    humidity = x[1]
    humidity = humidity.replace("Humidity=","")
    pressure = x[2]
    pressure = pressure.replace("Pressure=","")
    altitude = x[3]
    altitude = altitude.replace("Altitude=","")
    ts = time.time()
    reportTime = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
    location = "300 E main street"
```

def messagePersistence(self, client, userdata, message):

```
    data = message.payload
    data = str(data)
    data = data[3:-2]
    print(data)
    x = data.split(" ")
    temp = x[0]
    temp = temp.replace("Temp=","")
    humidity = x[1]
    humidity = humidity.replace("Humidity=","")
    pressure = x[2]
    pressure = pressure.replace("Pressure=","")
    altitude = x[3]
    altitude = altitude.replace("Altitude=","")
    ts = time.time()
    reportTime = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
    location = "300 E main street"
```

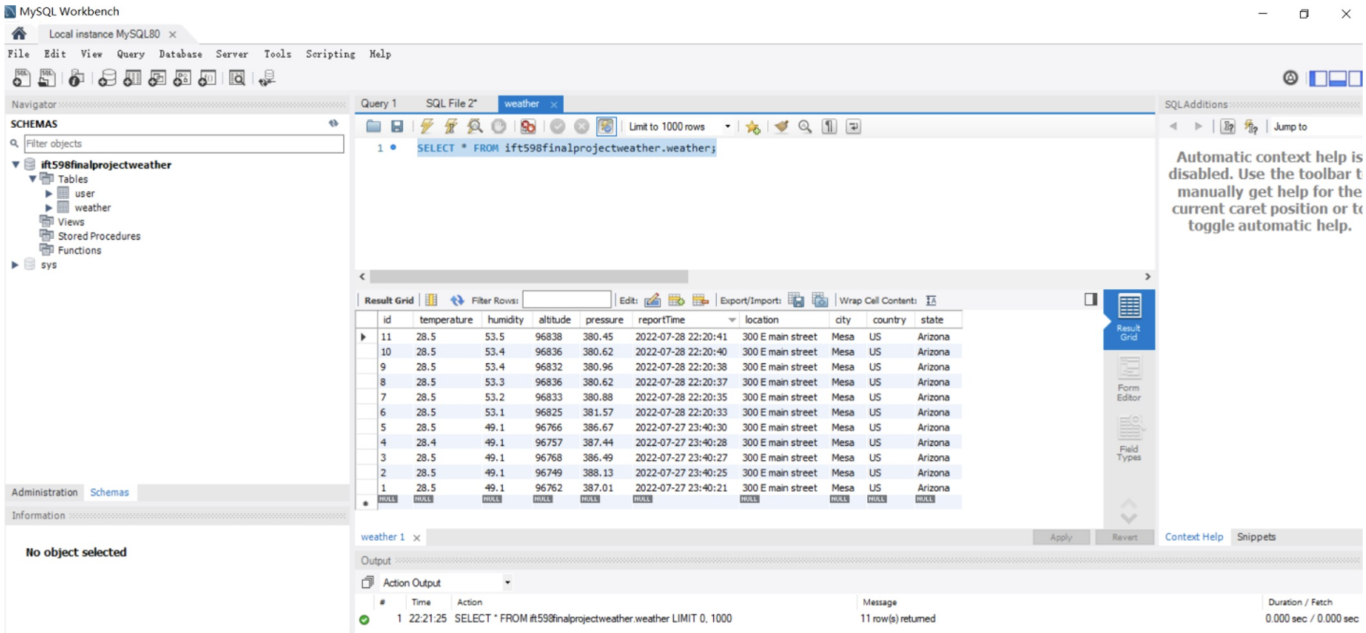
Then, you can use the below code to store data into MySQL database.



```
try:
    connection = mysql.connector.connect(host='localhost',
                                         database='ift598finalprojectweather',
                                         user='root',
                                         password='')

    cursor = connection.cursor()
    query = "INSERT INTO weather(temperature,
humidity,altitude,pressure,reportTime,location,city,country,state) " \
            "VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s)"
    args = (temp, humidity, pressure, altitude, reportTime,location,city,country,state)
    cursor.execute(query, args)
    connection.commit()

except mysql.connector.Error as e:
    print("Error reading data from MySQL table", e)
finally:
    if connection.is_connected():
        connection.close()
        cursor.close()
```

```
try:
    connection = mysql.connector.connect(host='localhost',
        database='ift598finalprojectweather',
        user='root',
        password='')

    cursor = connection.cursor()
    query = "INSERT INTO weather(temperature,
humidity, altitude, pressure, reportTime, location, city, country, state) " \
        "VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s)"
    args = (temp, humidity, pressure, altitude, reportTime, location, city, country, state)
    cursor.execute(query, args)
    connection.commit()

except mysql.connector.Error as e:
    print("Error reading data from MySQL table", e)
finally:
    if connection.is_connected():
        connection.close()
        cursor.close()
```

When you are successful, you can see the data in the weather table as below figure 15. Appendix includes Create weather statement.



```
SELECT * FROM ift598finalprojectweather.weather;
```

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

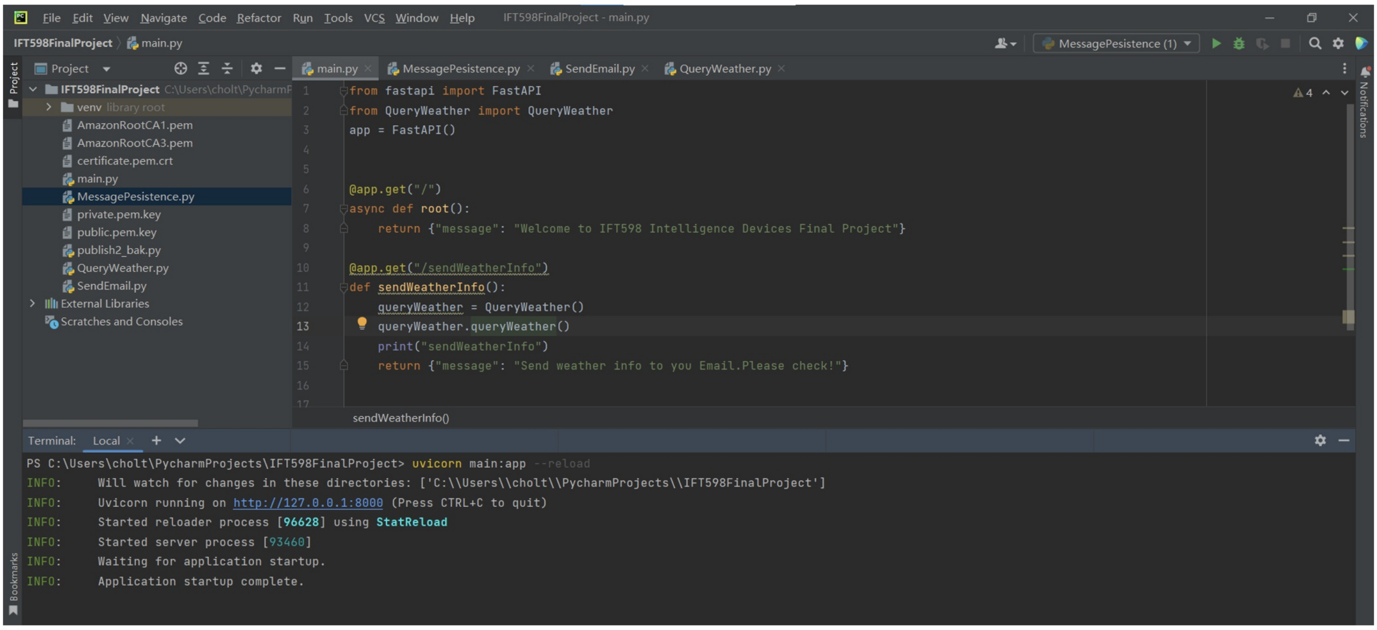
No object selected

id	temperature	humidity	altitude	pressure	reportTime	location	city	country	state
11	28.5	53.5	96838	380.45	2022-07-28 22:20:41	300 E main street	Mesa	US	Arizona
10	28.5	53.4	96836	380.62	2022-07-28 22:20:40	300 E main street	Mesa	US	Arizona
9	28.5	53.4	96832	380.96	2022-07-28 22:20:38	300 E main street	Mesa	US	Arizona
8	28.5	53.3	96836	380.62	2022-07-28 22:20:37	300 E main street	Mesa	US	Arizona
7	28.5	53.2	96833	380.88	2022-07-28 22:20:35	300 E main street	Mesa	US	Arizona
6	28.5	53.1	96825	381.57	2022-07-28 22:20:33	300 E main street	Mesa	US	Arizona
5	28.5	49.1	96766	386.67	2022-07-27 23:40:30	300 E main street	Mesa	US	Arizona
4	28.4	49.1	96757	387.44	2022-07-27 23:40:28	300 E main street	Mesa	US	Arizona
3	28.5	49.1	96768	386.49	2022-07-27 23:40:27	300 E main street	Mesa	US	Arizona
2	28.5	49.1	96749	388.13	2022-07-27 23:40:25	300 E main street	Mesa	US	Arizona
1	28.5	49.1	96762	387.01	2022-07-27 23:40:21	300 E main street	Mesa	US	Arizona
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
#   Time      Action                                                   Message              Duration / Fetch
1   22:21:25  SELECT * FROM ift598finalprojectweather.weather LIMIT 0, 1000   11 row(s) returned   0.000 sec / 0.000 sec
```

Figure 15.

In the figure below, you need to install fastAPI as a restful server. If you install successfully, you can use command: uvicorn main:app to start the Restful server.

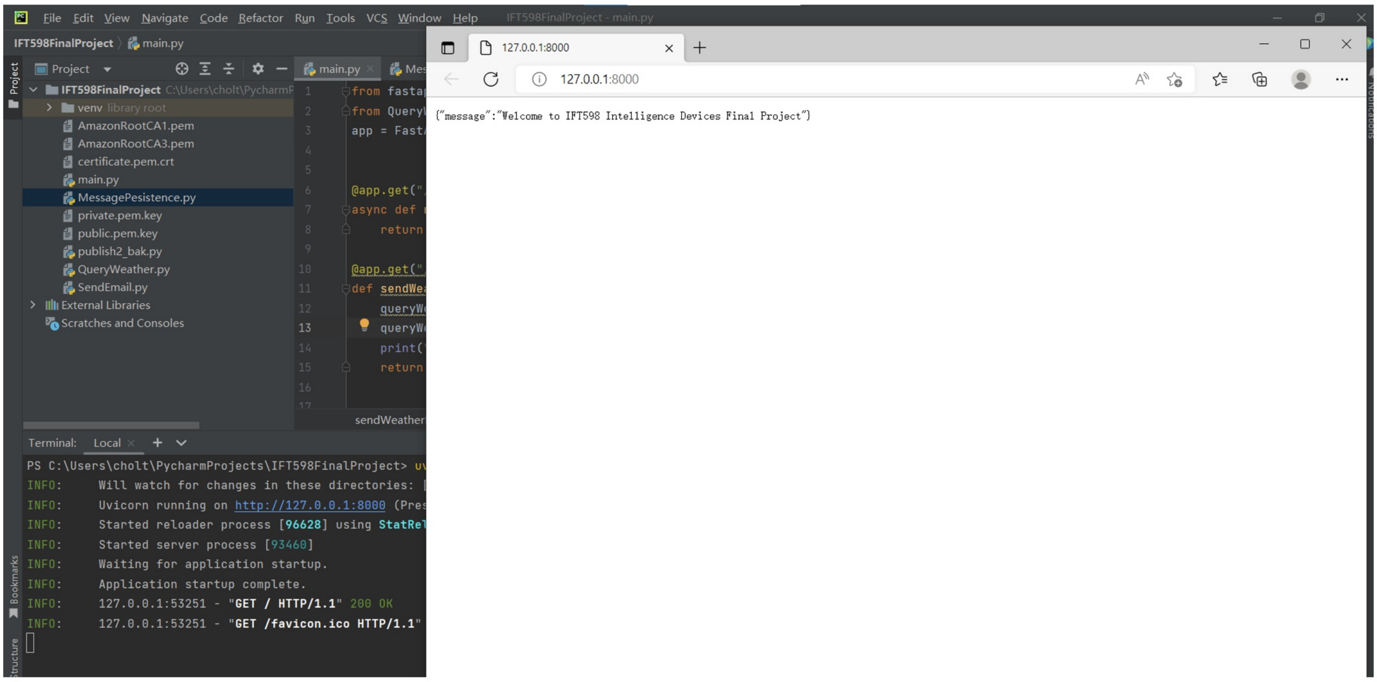


```
from fastapi import FastAPI
from QueryWeather import QueryWeather
app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Welcome to IFT598 Intelligence Devices Final Project"}

@app.get("/sendWeatherInfo")
def sendWeatherInfo():
    queryWeather = QueryWeather()
    queryWeather.queryWeather()
    print("sendWeatherInfo")
    return {"message": "Send weather info to you Email.Please check!"}
```

```
PS C:\Users\cholt\PycharmProjects\IFT598FinalProject> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\cholt\\PycharmProjects\\IFT598FinalProject']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [96628] using StatReload
INFO:     Started server process [93460]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Figure 16.

Open browser, Go to localhost:8000. You can see the index page as below figure 17.

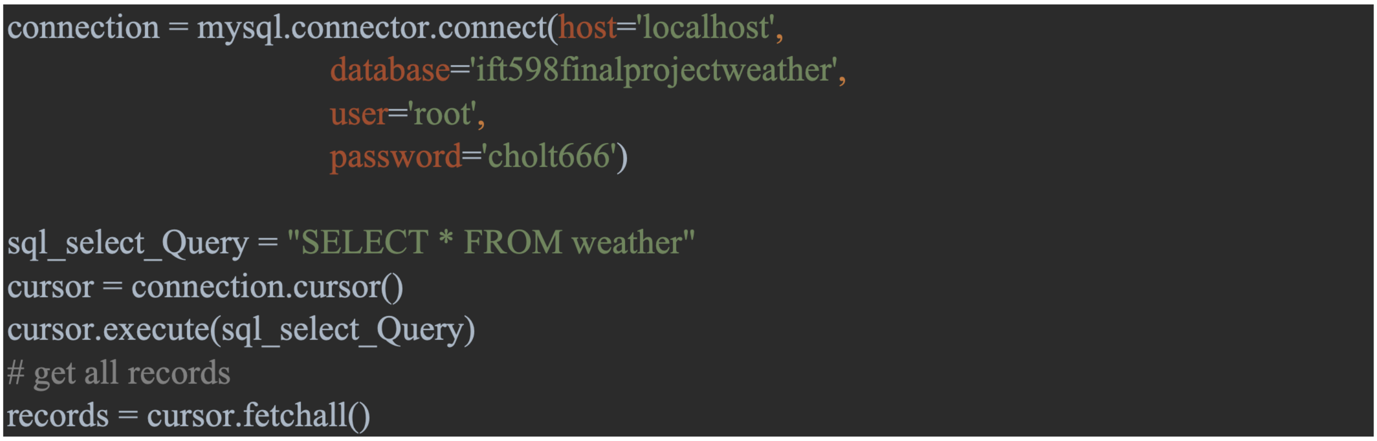


```
{"message":"Welcome to IFT598 Intelligence Devices Final Project"}
```

```
PS C:\Users\cholt\PycharmProjects\IFT598FinalProject> uv
INFO:     Will watch for changes in these directories: [
INFO:     Uvicorn running on http://127.0.0.1:8000 (Pres
INFO:     Started reloader process [96628] using StatRel
INFO:     Started server process [93460]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:53251 - "GET / HTTP/1.1" 200 OK
INFO:     127.0.0.1:53251 - "GET /favicon.ico HTTP/1.1"
```

Figure 17.

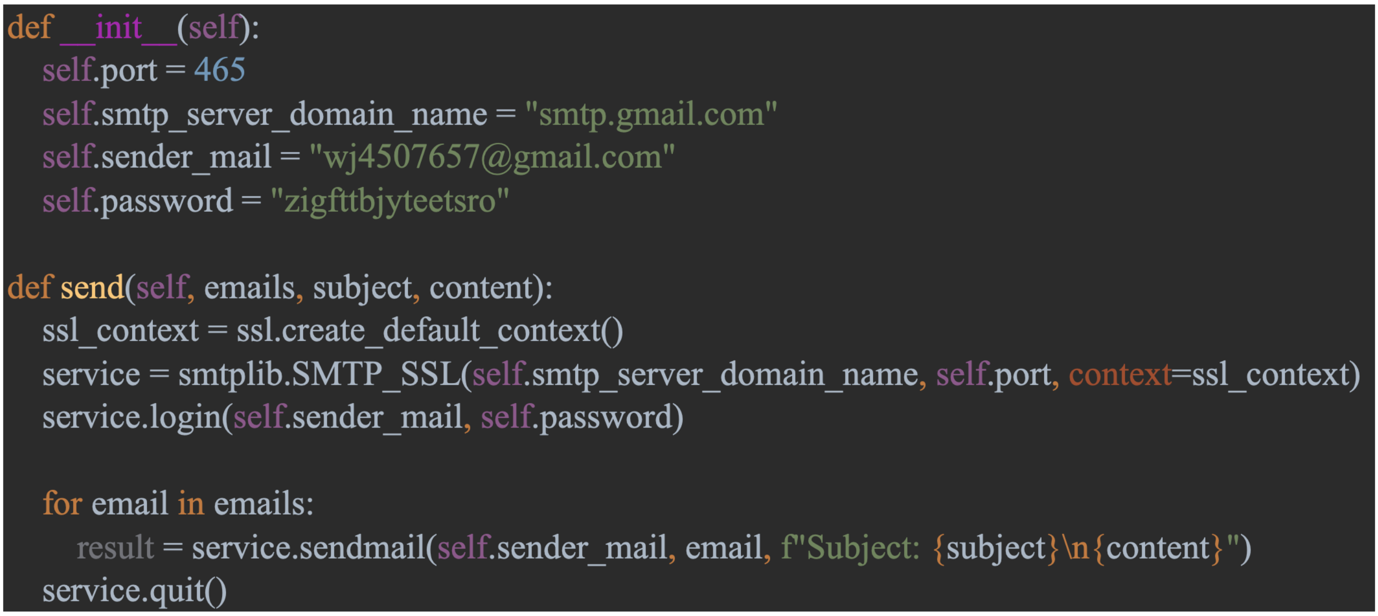
To send email to users, first we need to query data from MySQL, the below code will do this.



```
connection = mysql.connector.connect(host='localhost',
                                     database='ift598finalprojectweather',
                                     user='root',
                                     password='cholt666')

sql_select_Query = "SELECT * FROM weather"
cursor = connection.cursor()
cursor.execute(sql_select_Query)
# get all records
records = cursor.fetchall()
```

```
connection = mysql.connector.connect(host='localhost',
    database='ift598finalprojectweather',
    user='root',
    password='cholt666')sql_select_Query = "SELECT * FROM weather"
cursor = connection.cursor()
cursor.execute(sql_select_Query)
# get all records
records = cursor.fetchall()
```

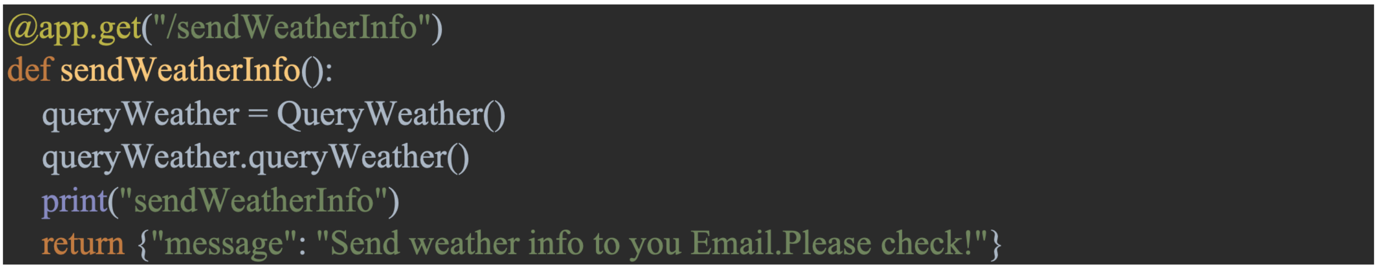
And then, we can invoke the send email function as below code:



```
def __init__(self):
    self.port = 465
    self.smtp_server_domain_name = "smtp.gmail.com"
    self.sender_mail = "wj4507657@gmail.com"
    self.password = "zigfttbjyteetsro"

def send(self, emails, subject, content):
    ssl_context = ssl.create_default_context()
    service = smtplib.SMTP_SSL(self.smtp_server_domain_name, self.port, context=ssl_context)
    service.login(self.sender_mail, self.password)

    for email in emails:
        result = service.sendmail(self.sender_mail, email, f"Subject: {subject}\n{content}")
    service.quit()
```

```
def __init__(self):
  self.port = 465
  self.smtp_server_domain_name = "smtp.gmail.com"
  self.sender_mail = "wj4507657@gmail.com"
  self.password = "zigfttbjyteetsro"


def send(self, emails, subject, content):
    ssl_context = ssl.create_default_context()
    service = smtplib.SMTP_SSL(self.smtp_server_domain_name, self.port, context=ssl_context)
    service.login(self.sender_mail, self.password)

    for email in emails:
        result = service.sendmail(self.sender_mail, email, f"Subject: {subject}\n{content}")
    service.quit()
```

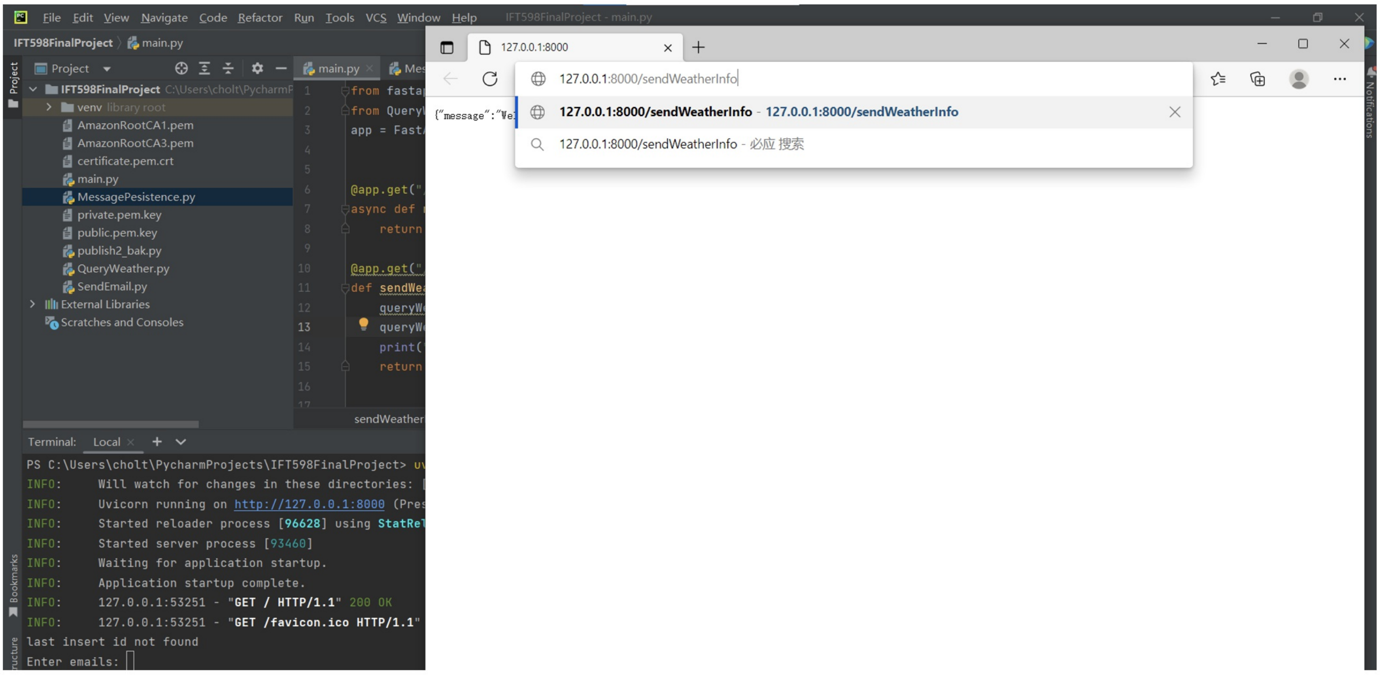
The restful server code as below:



```
@app.get("/sendWeatherInfo")
def sendWeatherInfo():
    queryWeather = QueryWeather()
    queryWeather.queryWeather()
    print("sendWeatherInfo")
    return {"message": "Send weather info to you Email.Please check!"}
```

```
@app.get("/sendWeatherInfo")
def sendWeatherInfo():
    queryWeather = QueryWeather()
    queryWeather.queryWeather()
    print("sendWeatherInfo")
    return {"message": "Send weather info to you Email.Please check!"}
```

Open browser, Go to localhost:8000/sendWeatherInfo. You can see the return page as below figure 18.



File Edit View Navigate Code Refactor Run Tools VCS Window Help
IFT598FinalProject - main.py
IFT598FinalProject 〉 main.py
Project
IFT598FinalProject C:\Users\cholt\PycharmP
venv library root
AmazonRootCA1.pem
AmazonRootCA3.pem
certificate.pem.crt
main.py
MessagePesistence.py
private.pem.key
public.pem.key
publish2_bak.py
QueryWeather.py
SendEmail.py
External Libraries
Scratches and Consoles
main.py
Mes
sendWeather
Terminal: Local
127.0.0.1:8000
127.0.0.1:8000/sendWeatherInfo
127.0.0.1:8000/sendWeatherInfo - 127.0.0.1:8000/sendWeatherInfo
127.0.0.1:8000/sendWeatherInfo - 必应 搜索
Bookmarks
Notifications

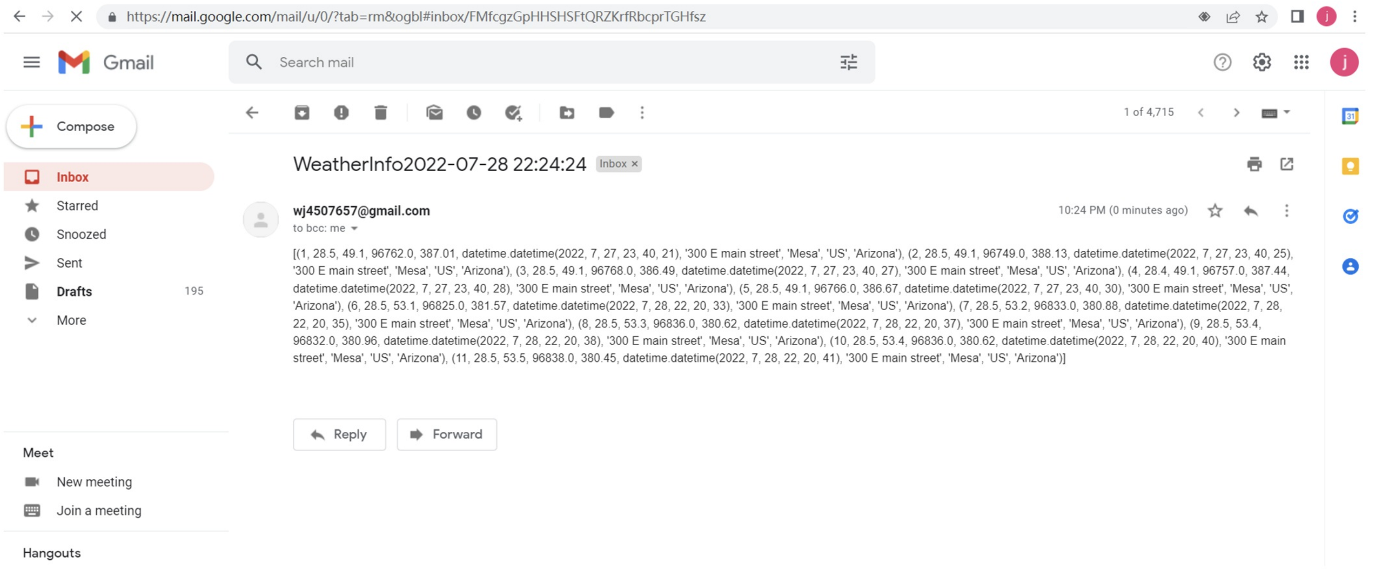
```
from fasta
from Query
app = Fast


@app.get("
async def 
    return

@app.get("
def sendWea
    queryWe
    queryWe
    print(
    return
```

```
{"message":"We
```

```
PS C:\Users\cholt\PycharmProjects\IFT598FinalProject> uv
INFO:     Will watch for changes in these directories: [
INFO:     Uvicorn running on http://127.0.0.1:8000 (Pres
INFO:     Started reloader process [96628] using StatRel
INFO:     Started server process [93460]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:53251 - "GET / HTTP/1.1" 200 OK
INFO:     127.0.0.1:53251 - "GET /favicon.ico HTTP/1.1"
last insert id not found
Enter emails: 
```

Figure 18.

For testing whether the email was sent successfully or not, open your Gmail account, you will see the email notification as below Figure 19.



WeatherInfo2022-07-28 22:24:24

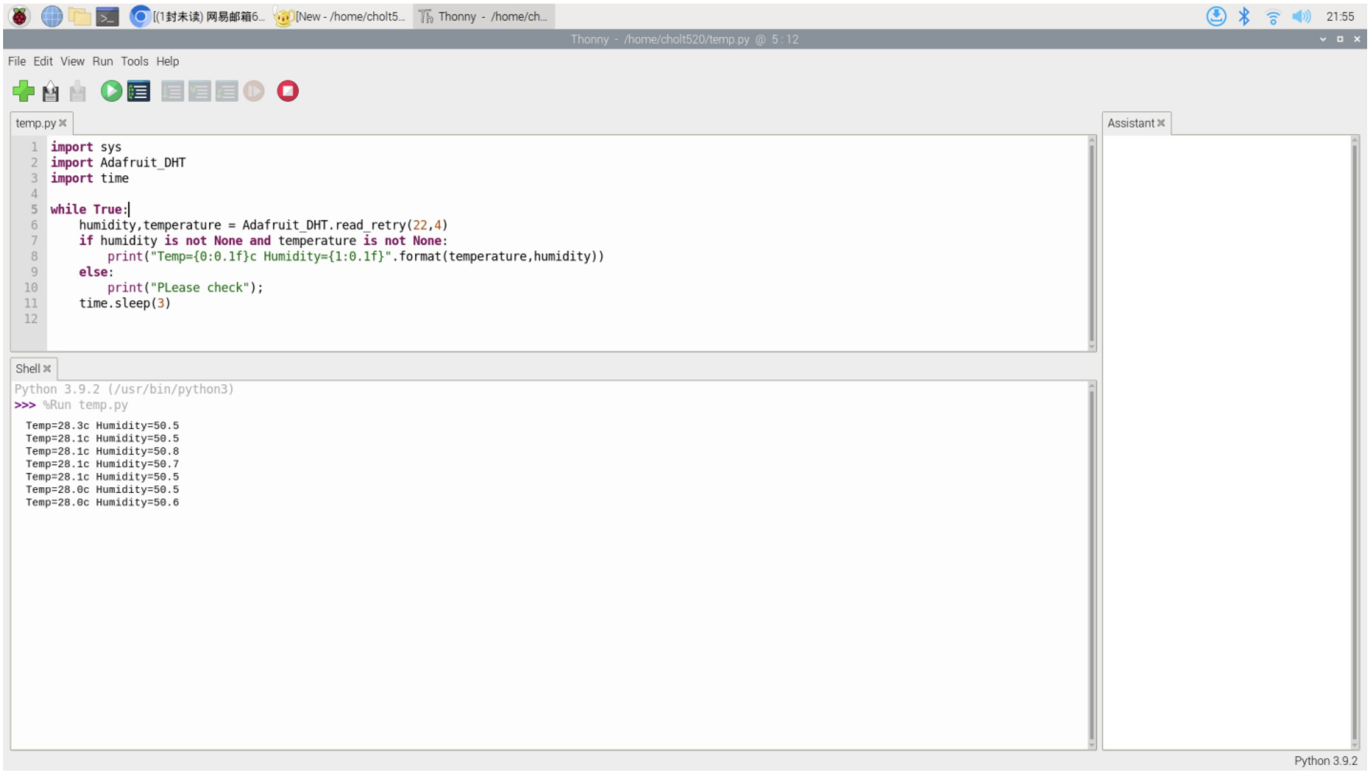
wj4507657@gmail.com
to bcc: me

```
[(1, 28.5, 49.1, 96762.0, 387.01, datetime.datetime(2022, 7, 27, 23, 40, 21), '300 E main street', 'Mesa', 'US', 'Arizona'), (2, 28.5, 49.1, 96749.0, 388.13, datetime.datetime(2022, 7, 27, 23, 40, 25), '300 E main street', 'Mesa', 'US', 'Arizona'), (3, 28.5, 49.1, 96768.0, 386.49, datetime.datetime(2022, 7, 27, 23, 40, 27), '300 E main street', 'Mesa', 'US', 'Arizona'), (4, 28.4, 49.1, 96757.0, 387.44, datetime.datetime(2022, 7, 27, 23, 40, 28), '300 E main street', 'Mesa', 'US', 'Arizona'), (5, 28.5, 49.1, 96766.0, 386.67, datetime.datetime(2022, 7, 27, 23, 40, 30), '300 E main street', 'Mesa', 'US', 'Arizona'), (6, 28.5, 53.1, 96825.0, 381.57, datetime.datetime(2022, 7, 28, 22, 20, 33), '300 E main street', 'Mesa', 'US', 'Arizona'), (7, 28.5, 53.2, 96833.0, 380.88, datetime.datetime(2022, 7, 28, 22, 20, 35), '300 E main street', 'Mesa', 'US', 'Arizona'), (8, 28.5, 53.3, 96836.0, 380.62, datetime.datetime(2022, 7, 28, 22, 20, 37), '300 E main street', 'Mesa', 'US', 'Arizona'), (9, 28.5, 53.4, 96832.0, 380.96, datetime.datetime(2022, 7, 28, 22, 20, 38), '300 E main street', 'Mesa', 'US', 'Arizona'), (10, 28.5, 53.4, 96836.0, 380.62, datetime.datetime(2022, 7, 28, 22, 20, 40), '300 E main street', 'Mesa', 'US', 'Arizona'), (11, 28.5, 53.5, 96838.0, 380.45, datetime.datetime(2022, 7, 28, 22, 20, 41), '300 E main street', 'Mesa', 'US', 'Arizona')]
```

Figure 19.

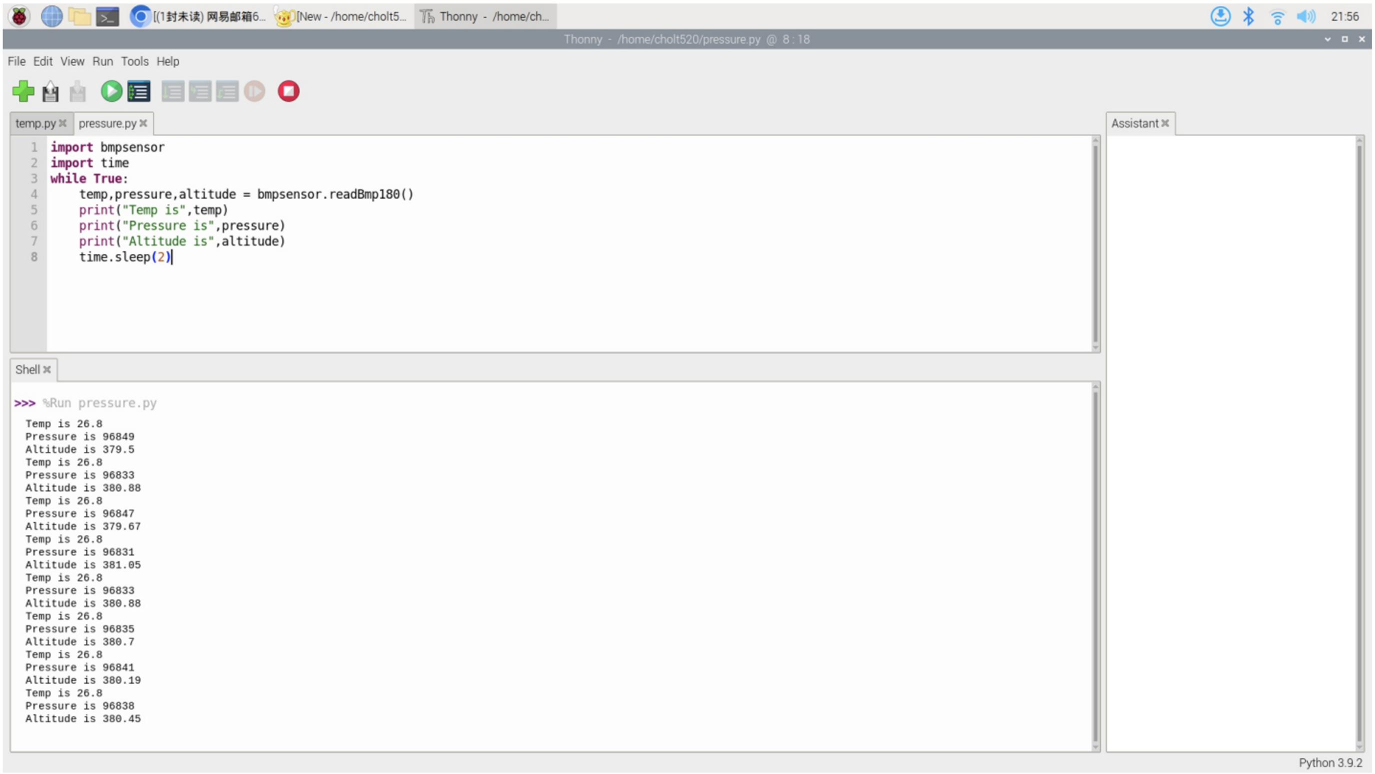
Testing and Result

For test DHT22 and BMP180, the below code can be used as the test part. The sample way to test temperature and humidity change. You can blow your breath to DHT22, as a result, you will see the change immediately.



```
import sys
import Adafruit_DHT
import time

while True:
    humidity,temperature = Adafruit_DHT.read_retry(22,4)
    if humidity is not None and temperature is not None:
        print("Temp={0:0.1f}c Humidity={1:0.1f}".format(temperature,humidity))
    else:
        print("PLease check");
    time.sleep(3)
```

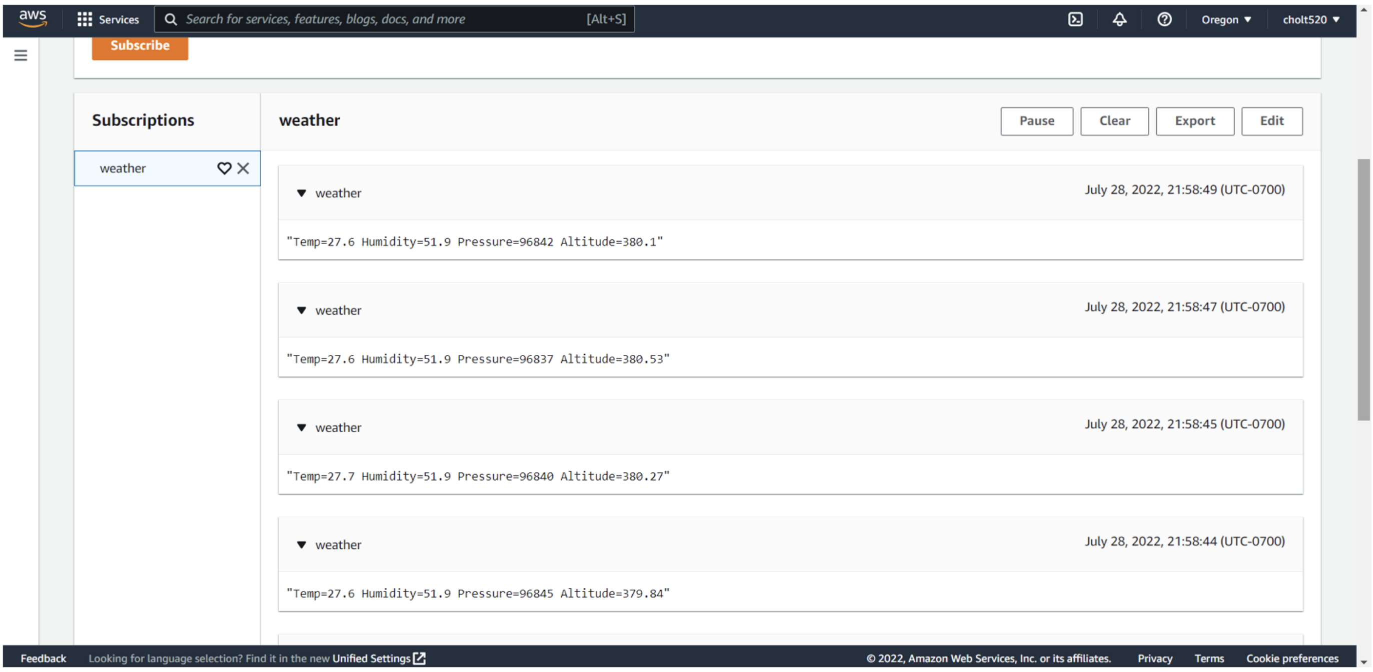
```
Python 3.9.2 (/usr/bin/python3)
>>> %Run temp.py
 Temp=28.3c Humidity=50.5
 Temp=28.1c Humidity=50.5
 Temp=28.1c Humidity=50.8
 Temp=28.1c Humidity=50.7
 Temp=28.1c Humidity=50.5
 Temp=28.0c Humidity=50.5
 Temp=28.0c Humidity=50.6
```



```
import bmpsensor
import time
while True:
    temp,pressure,altitude = bmpsensor.readBmp180()
    print("Temp is",temp)
    print("Pressure is",pressure)
    print("Altitude is",altitude)
    time.sleep(2)
```

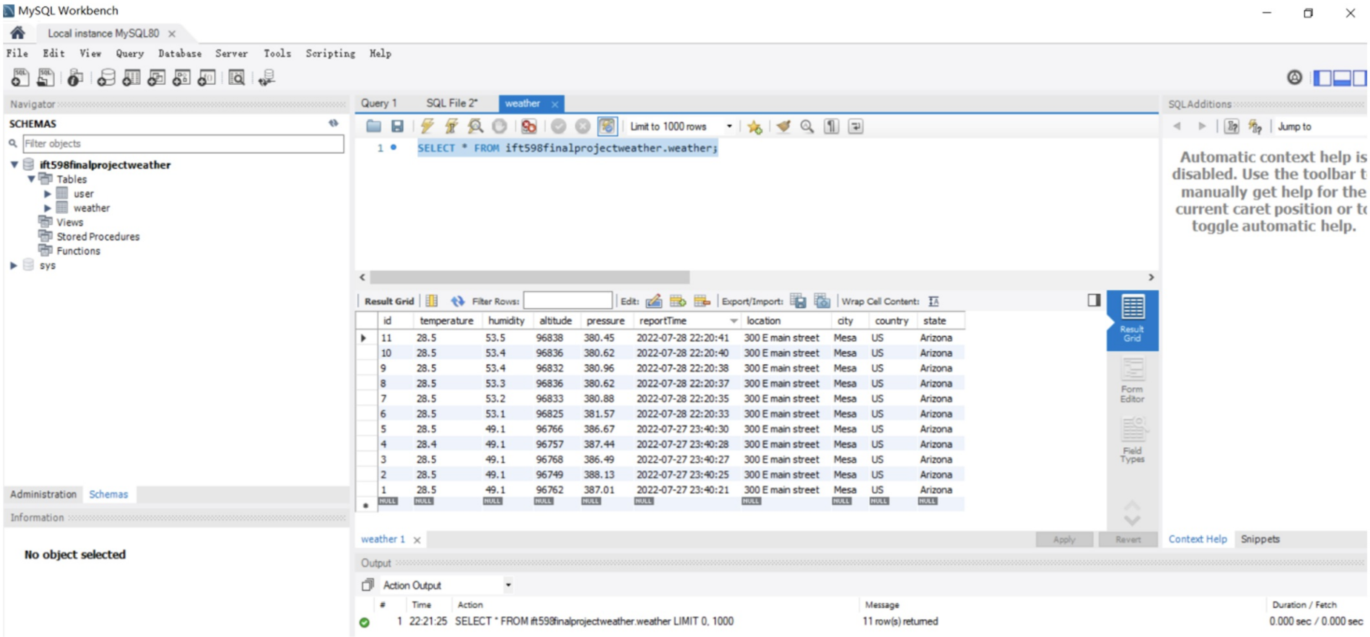
```
>>> %Run pressure.py
 Temp is 26.8
 Pressure is 96849
 Altitude is 379.5
 Temp is 26.8
 Pressure is 96833
 Altitude is 380.88
 Temp is 26.8
 Pressure is 96847
 Altitude is 379.67
 Temp is 26.8
 Pressure is 96831
 Altitude is 381.05
 Temp is 26.8
 Pressure is 96833
 Altitude is 380.88
 Temp is 26.8
 Pressure is 96835
 Altitude is 380.7
 Temp is 26.8
 Pressure is 96841
 Altitude is 380.19
 Temp is 26.8
 Pressure is 96838
 Altitude is 380.45
```

Login into AWS MQTT console, subscribe to the topic. You can test the data whether it is transfer successfully.


aws
Services
Search for services, features, blogs, docs, and more [Alt+S]
Oregon
cholt520
Subscribe
Subscriptions
weather
weather
Pause
Clear
Export
Edit
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

```
▼ weather                                   July 28, 2022, 21:58:49 (UTC-0700)
"Temp=27.6 Humidity=51.9 Pressure=96842 Altitude=380.1"

▼ weather                                   July 28, 2022, 21:58:47 (UTC-0700)
"Temp=27.6 Humidity=51.9 Pressure=96837 Altitude=380.53"

▼ weather                                   July 28, 2022, 21:58:45 (UTC-0700)
"Temp=27.7 Humidity=51.9 Pressure=96840 Altitude=380.27"

▼ weather                                   July 28, 2022, 21:58:44 (UTC-0700)
"Temp=27.6 Humidity=51.9 Pressure=96845 Altitude=379.84"
```

Using MySQL Workbench can easily see the data whether it is stored successfully.

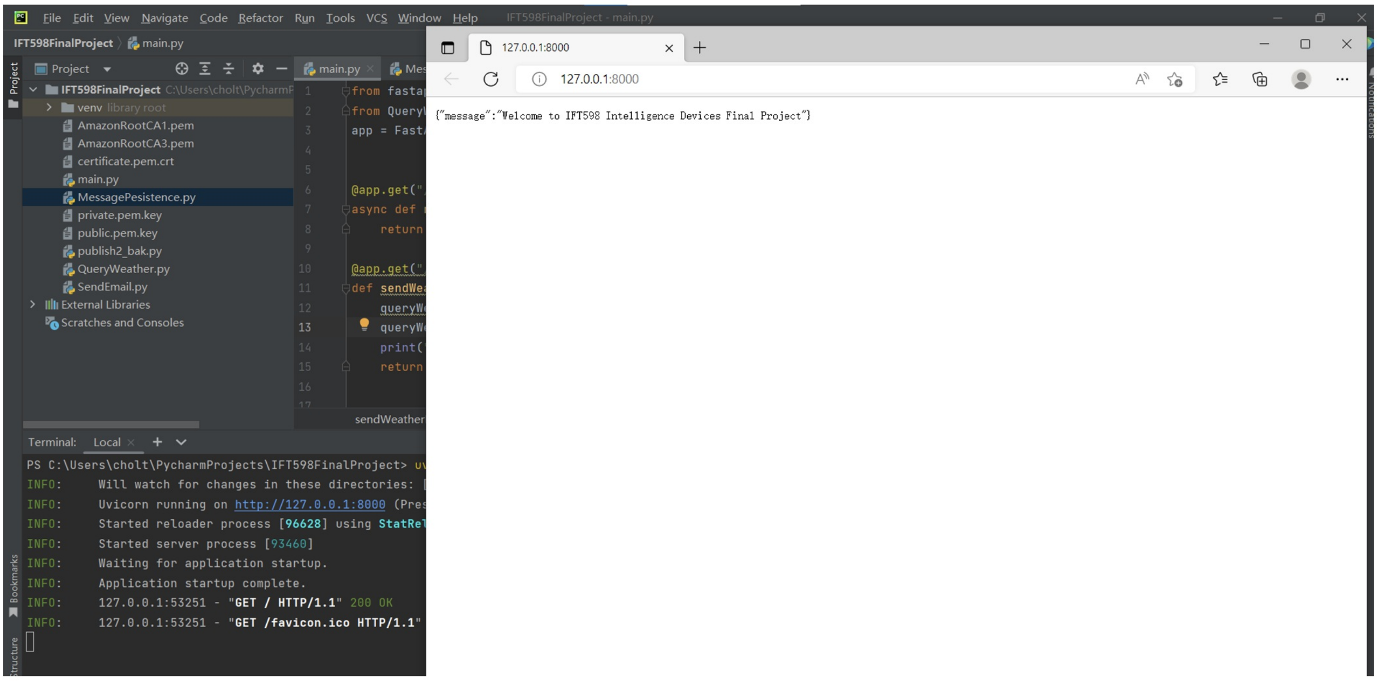


```sql
SELECT * FROM ift598finalprojectweather.weather;
```

id	temperature	humidity	altitude	pressure	reportTime	location	city	country	state
11	28.5	53.5	96838	380.45	2022-07-28 22:20:41	300 E main street	Mesa	US	Arizona
10	28.5	53.4	96836	380.62	2022-07-28 22:20:40	300 E main street	Mesa	US	Arizona
9	28.5	53.4	96832	380.96	2022-07-28 22:20:38	300 E main street	Mesa	US	Arizona
8	28.5	53.3	96836	380.62	2022-07-28 22:20:37	300 E main street	Mesa	US	Arizona
7	28.5	53.2	96833	380.88	2022-07-28 22:20:35	300 E main street	Mesa	US	Arizona
6	28.5	53.1	96825	381.57	2022-07-28 22:20:33	300 E main street	Mesa	US	Arizona
5	28.5	49.1	96766	386.67	2022-07-27 23:40:30	300 E main street	Mesa	US	Arizona
4	28.4	49.1	96757	387.44	2022-07-27 23:40:28	300 E main street	Mesa	US	Arizona
3	28.5	49.1	96768	386.49	2022-07-27 23:40:27	300 E main street	Mesa	US	Arizona
2	28.5	49.1	96749	388.13	2022-07-27 23:40:25	300 E main street	Mesa	US	Arizona
1	28.5	49.1	96762	387.01	2022-07-27 23:40:21	300 E main street	Mesa	US	Arizona
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                                   Message               Duration / Fetch
1  22:21:25  SELECT * FROM ift598finalprojectweather.weather LIMIT 0, 1000   11 row(s) returned   0.000 sec / 0.000 sec
```

Open browser, Go to localhost:8000. It can test the restful server whether it is installed successfully.

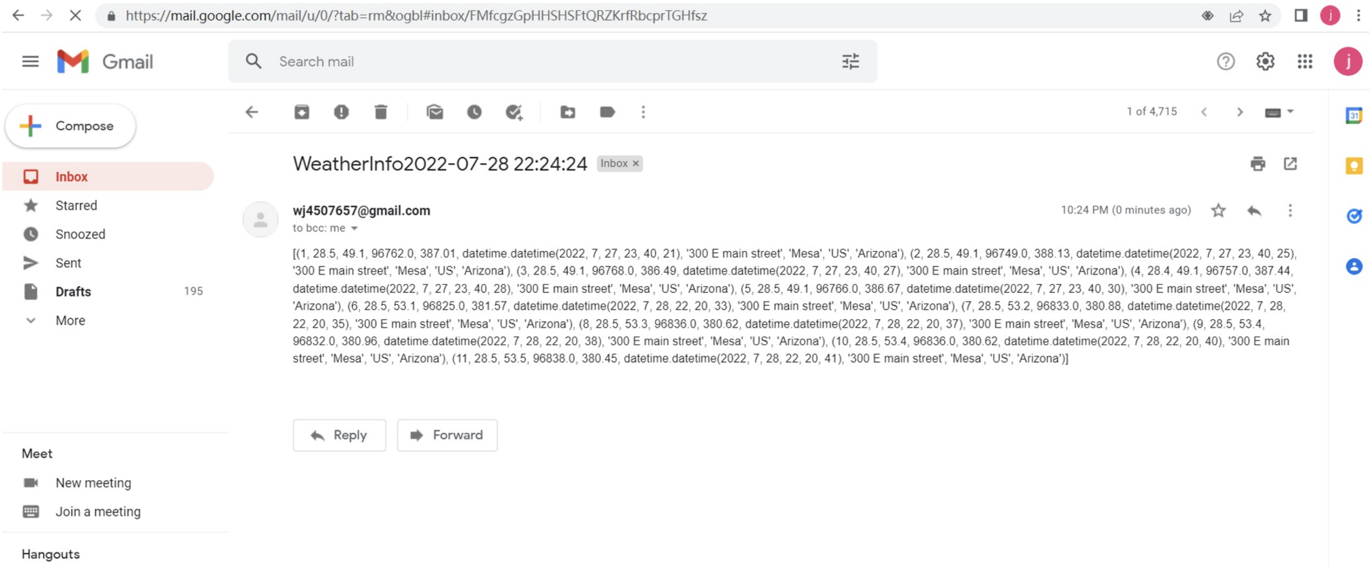


```
from fasta
from Query
app = Fast

@app.get("
async def 
    return

@app.get("
def sendWea
    queryWe
    queryWe
    print(
    return
```

```
{"message":"Welcome to IFT598 Intelligence Devices Final Project"}
```

```
PS C:\Users\cholt\PycharmProjects\IFT598FinalProject> uv
INFO:     Will watch for changes in these directories: [
INFO:     Uvicorn running on http://127.0.0.1:8000 (Pres
INFO:     Started reloader process [96628] using StatRel
INFO:     Started server process [93460]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:53251 - "GET / HTTP/1.1" 200 OK
INFO:     127.0.0.1:53251 - "GET /favicon.ico HTTP/1.1"
```

Open your Gmail account, you will see the email notification which test the email was sent successfully or not.



WeatherInfo2022-07-28 22:24:24

wj4507657@gmail.com

to bcc: me

10:24 PM (0 minutes ago)

[(1, 28.5, 49.1, 96762.0, 387.01, datetime.datetime(2022, 7, 27, 23, 40, 21), '300 E main street', 'Mesa', 'US', 'Arizona'), (2, 28.5, 49.1, 96749.0, 388.13, datetime.datetime(2022, 7, 27, 23, 40, 25), '300 E main street', 'Mesa', 'US', 'Arizona'), (3, 28.5, 49.1, 96768.0, 386.49, datetime.datetime(2022, 7, 27, 23, 40, 27), '300 E main street', 'Mesa', 'US', 'Arizona'), (4, 28.4, 49.1, 96757.0, 387.44, datetime.datetime(2022, 7, 27, 23, 40, 28), '300 E main street', 'Mesa', 'US', 'Arizona'), (5, 28.5, 49.1, 96766.0, 386.67, datetime.datetime(2022, 7, 27, 23, 40, 30), '300 E main street', 'Mesa', 'US', 'Arizona'), (6, 28.5, 53.1, 96825.0, 381.57, datetime.datetime(2022, 7, 28, 22, 20, 33), '300 E main street', 'Mesa', 'US', 'Arizona'), (7, 28.5, 53.2, 96833.0, 380.88, datetime.datetime(2022, 7, 28, 22, 20, 35), '300 E main street', 'Mesa', 'US', 'Arizona'), (8, 28.5, 53.3, 96836.0, 380.62, datetime.datetime(2022, 7, 28, 22, 20, 37), '300 E main street', 'Mesa', 'US', 'Arizona'), (9, 28.5, 53.4, 96832.0, 380.96, datetime.datetime(2022, 7, 28, 22, 20, 38), '300 E main street', 'Mesa', 'US', 'Arizona'), (10, 28.5, 53.4, 96836.0, 380.62, datetime.datetime(2022, 7, 28, 22, 20, 40), '300 E main street', 'Mesa', 'US', 'Arizona'), (11, 28.5, 53.5, 96838.0, 380.45, datetime.datetime(2022, 7, 28, 22, 20, 41), '300 E main street', 'Mesa', 'US', 'Arizona')]

Weather monitor system Integration testing:

To test the performance of the IoT weather monitoring system, sensors can be placed in various locations. For instance, the system can be set up indoors to measure temperature, humidity, and pressure. Initially, the system can be tested over a 24-hour period, then extended to one week, one month, and so on, to assess its long-term reliability.

Testing should also include evaluating the portable power supply. If deploying the IoT system in a remote field, it is essential to determine how long the system can operate without losing power and how long the portable power can sustain the system. Additionally, it is crucial to test the system's scalability by deploying hundreds or thousands of sensors to ensure it continues to function correctly under increased load.

The DHT22 sensor, with its wide temperature range of -40 to 80°C, can operate effectively in harsh environments, making it suitable for testing in various conditions.

Summary and Conclusion

This project is designed and developed for IoT weather monitoring, utilizing weather parameters such as temperature, humidity, pressure, and altitude. The IoT weather monitoring system collects real-time data and sends it to a server for analysis and notification purposes.

During the design process, a list of requirements will help readers understand what is needed for the system. Block diagrams and data flow charts provide a deeper understanding of the entire process. The project interface displays all parameters and functions to developers, aiding them in easily building their own IoT system.

The development process details the core code of the IoT system, explaining how to send data from sensors and receive data from RESTful servers. The testing and results section describes how to test the entire system to ensure it works properly and provides strategies for troubleshooting errors.

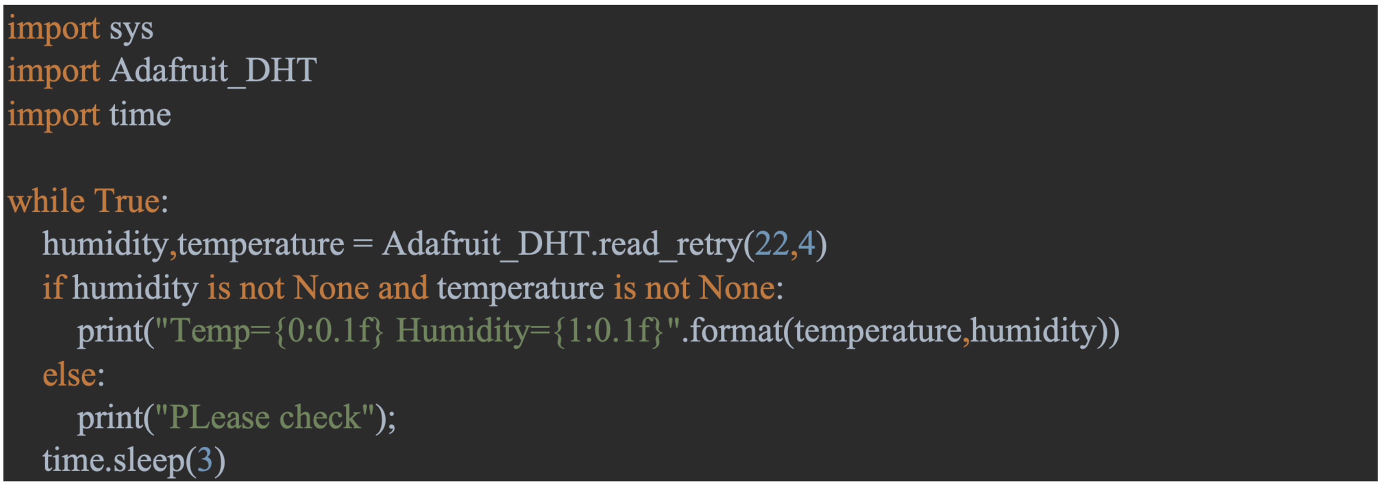
This IoT weather monitoring system is designed and developed for weather prediction and forest fire prevention. In California, accurate weather monitoring is crucial for residents' safety and preparedness.

Appendix

Table:

```
create table weather(
id int auto_increment primary key,
temperature double,
humidity double,
altitude double,
pressure double,
reportTime timestamp,
location varchar(32),
city varchar(32),
country varchar(32),
state varchar(32)
)
```

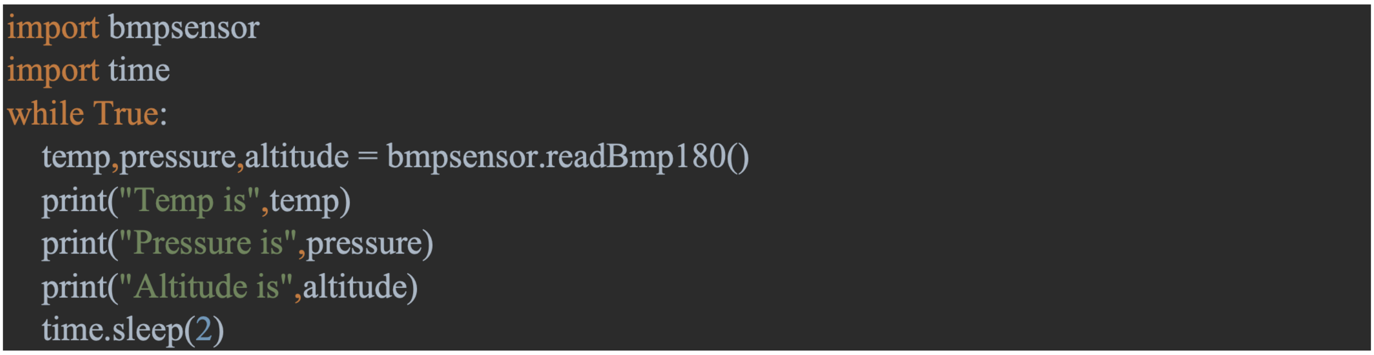
Temp.py



```
import sys
import Adafruit_DHT
import time

while True:
    humidity,temperature = Adafruit_DHT.read_retry(22,4)
    if humidity is not None and temperature is not None:
        print("Temp={0:0.1f} Humidity={1:0.1f}".format(temperature,humidity))
    else:
        print("PLease check");
    time.sleep(3)
```

```
import sys
import Adafruit_DHT
import time

while True:
    humidity, temperature = Adafruit_DHT.read_retry(22,4)
    if humidity is not None and temperature is not None:
        print("Temp={0:0.1f} Humidity={1:0.1f}".format(temperature, humidity))
    else:
        print("PLease check");
    time.sleep(3)
```

Pressure.py

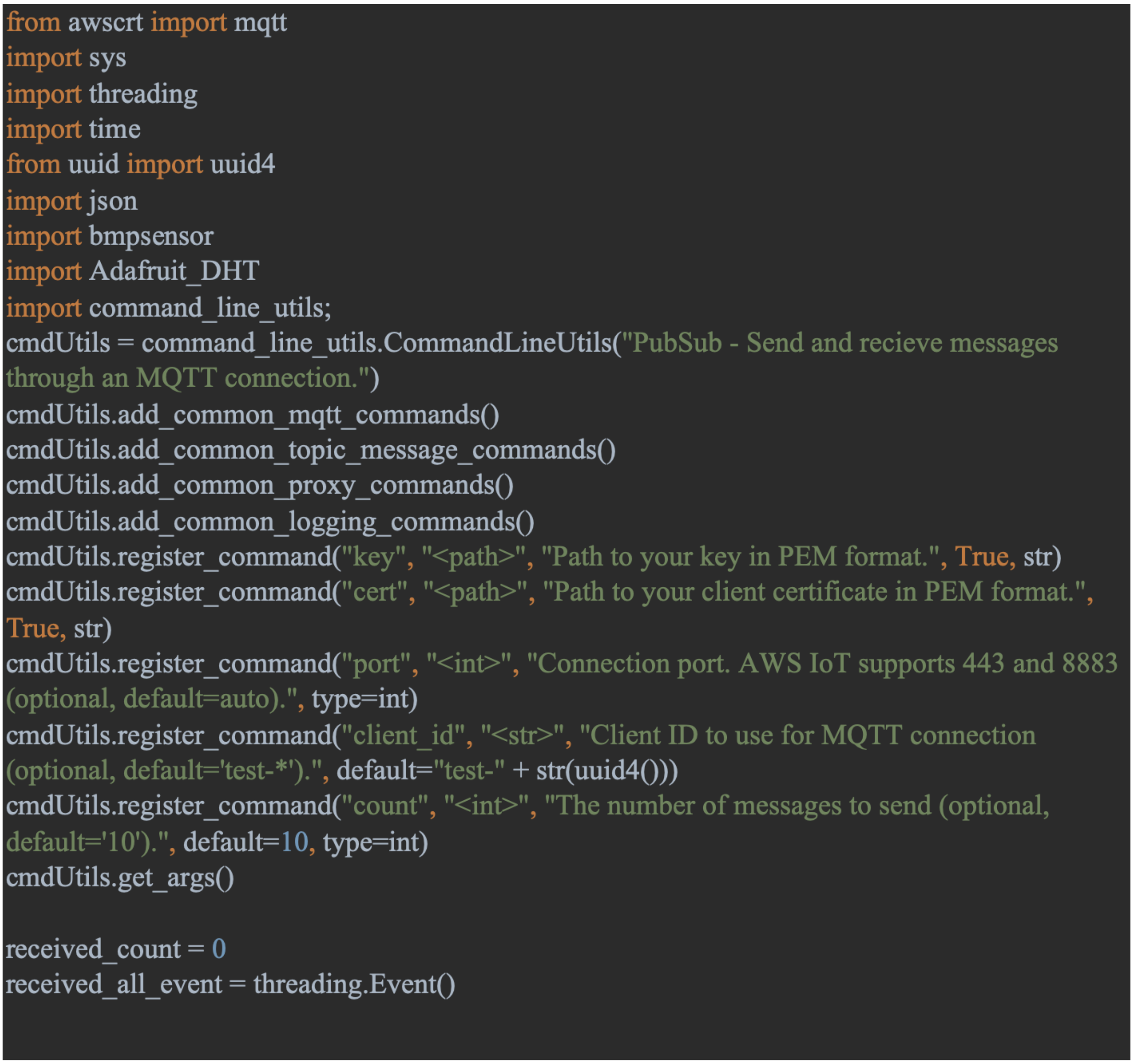


```
import bmpsensor
import time
while True:
    temp,pressure,altitude = bmpsensor.readBmp180()
    print("Temp is",temp)
    print("Pressure is",pressure)
    print("Altitude is",altitude)
    time.sleep(2)
```

```
import bmpsensor
import time
while True:
    temp, pressure, altitude = bmpsensor.readBmp180()
    print("Temp is", temp)
    print("Pressure is", pressure)
    print("Altitude is", altitude)
```

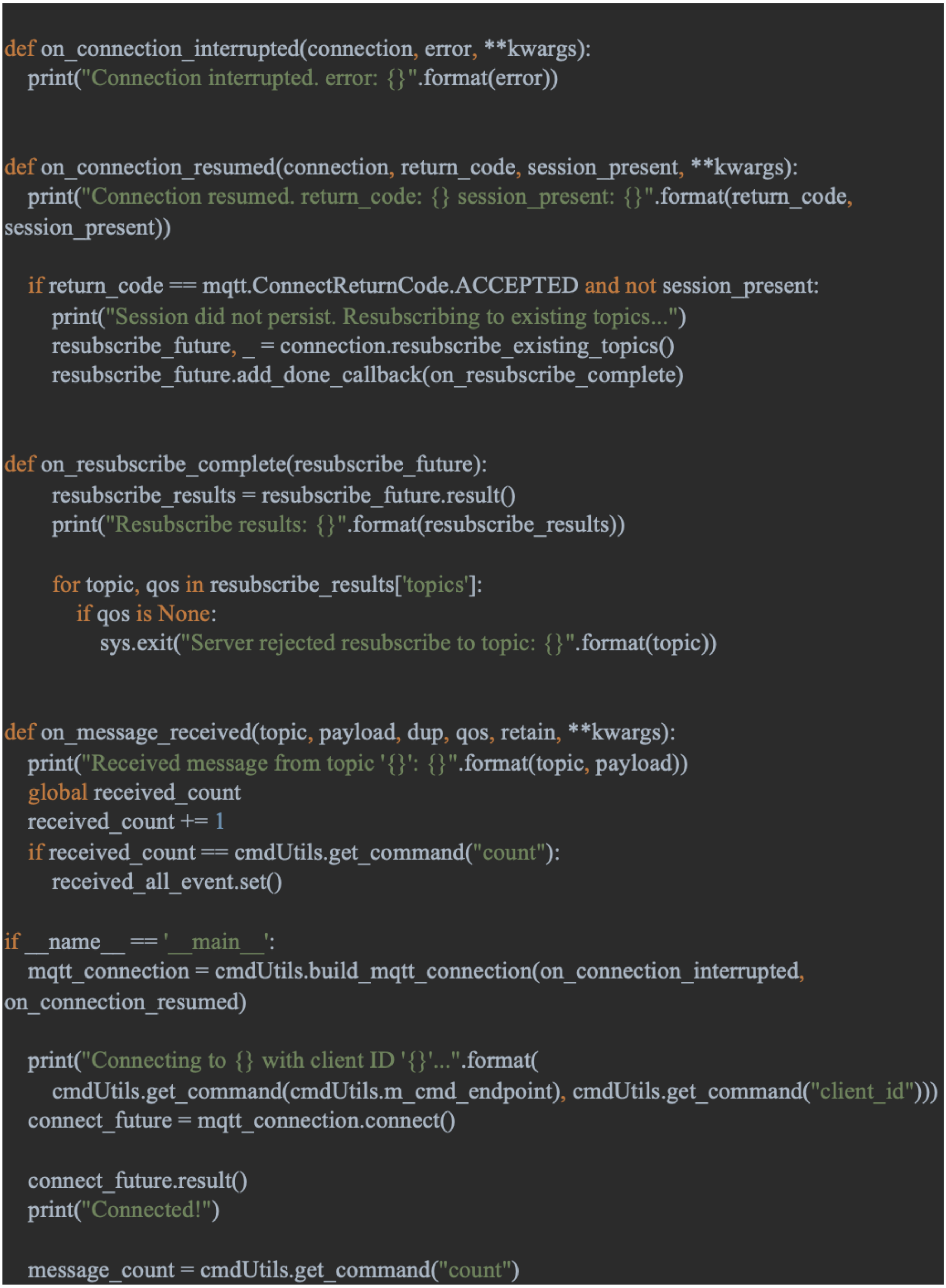
```
time.sleep(2)
```

Weather.py



```
from awscrt import mqtt
import sys
import threading
import time
from uuid import uuid4
import json
import bmpsensor
import Adafruit_DHT
import command_line_utils;
cmdUtils = command_line_utils.CommandLineUtils("PubSub - Send and recieve messages through an MQTT connection.")
cmdUtils.add_common_mqtt_commands()
cmdUtils.add_common_topic_message_commands()
cmdUtils.add_common_proxy_commands()
cmdUtils.add_common_logging_commands()
cmdUtils.register_command("key", "<path>", "Path to your key in PEM format.", True, str)
cmdUtils.register_command("cert", "<path>", "Path to your client certificate in PEM format.", True, str)
cmdUtils.register_command("port", "<int>", "Connection port. AWS IoT supports 443 and 8883 (optional, default=auto).", type=int)
cmdUtils.register_command("client_id", "<str>", "Client ID to use for MQTT connection (optional, default='test-*').", default="test-" + str(uuid4()))
cmdUtils.register_command("count", "<int>", "The number of messages to send (optional, default='10').", default=10, type=int)
cmdUtils.get_args()

received_count = 0
received_all_event = threading.Event()
```

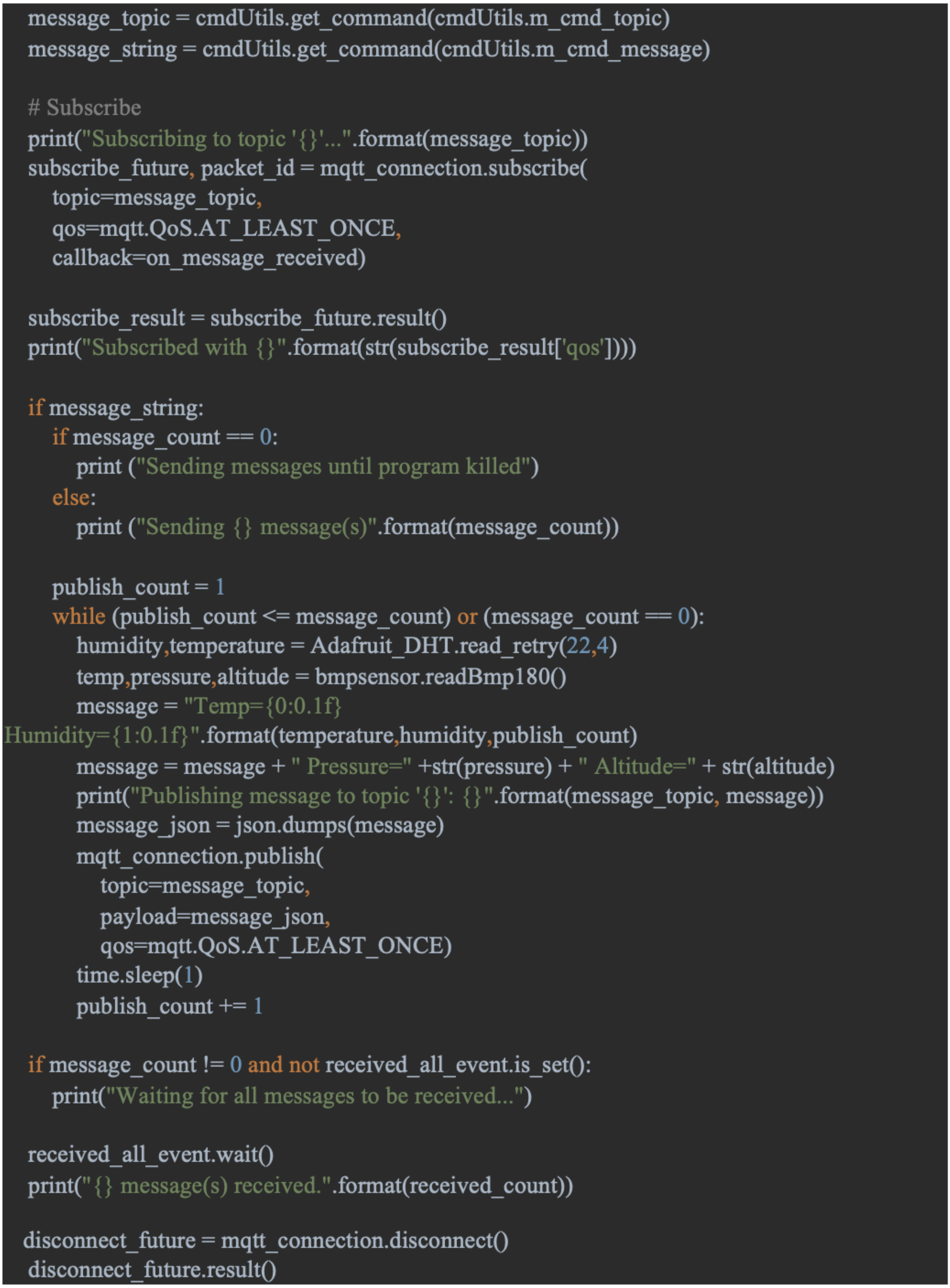
```
from awscrt import mqtt
import sys
import threading
import time
from uuid import uuid4
import json
import bmpsensor
import Adafruit_DHT
import command_line_utils;

cmdUtils = command_line_utils.CommandLineUtils("PubSub - Send and recieve messages through an MQTT connection.")

cmdUtils.add_common_mqtt_commands()

cmdUtils.add_common_topic_message_commands()

cmdUtils.add_common_proxy_commands()

cmdUtils.add_common_logging_commands()

cmdUtils.register_command("key", "<path>", "Path to your key in PEM format.", True, str)

cmdUtils.register_command("cert", "<path>", "Path to your client certificate in PEM format.", True, str)

cmdUtils.register_command("port", "<int>", "Connection port. AWS IoT supports 443 and 8883 (optional, default=auto).", type=int)

cmdUtils.register_command("client_id", "<str>", "Client ID to use for MQTT connection (optional, default='test-*').", default="test-" + str(uuid4()))

cmdUtils.register_command("count", "<int>", "The number of messages to send (optional, default='10').", default=10, type=int)

cmdUtils.get_args()received_count = 0

received_all_event = threading.Event()
```



```
def on_connection_interrupted(connection, error, **kwargs):
    print("Connection interrupted. error: {}".format(error))


def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print("Connection resumed. return_code: {} session_present: {}".format(return_code,
session_present))

    if return_code == mqtt.ConnectReturnCode.ACCEPTED and not session_present:
        print("Session did not persist. Resubscribing to existing topics...")
        resubscribe_future, _ = connection.resubscribe_existing_topics()
        resubscribe_future.add_done_callback(on_resubscribe_complete)


def on_resubscribe_complete(resubscribe_future):
        resubscribe_results = resubscribe_future.result()
        print("Resubscribe results: {}".format(resubscribe_results))

        for topic, qos in resubscribe_results['topics']:
            if qos is None:
                sys.exit("Server rejected resubscribe to topic: {}".format(topic))


def on_message_received(topic, payload, dup, qos, retain, **kwargs):
    print("Received message from topic '{}': {}".format(topic, payload))
    global received_count
    received_count += 1
    if received_count == cmdUtils.get_command("count"):
        received_all_event.set()

if __name__ == '__main__':
    mqtt_connection = cmdUtils.build_mqtt_connection(on_connection_interrupted,
on_connection_resumed)

    print("Connecting to {} with client ID '{}'...".format(
        cmdUtils.get_command(cmdUtils.m_cmd_endpoint), cmdUtils.get_command("client_id")))
    connect_future = mqtt_connection.connect()

    connect_future.result()
    print("Connected!")

    message_count = cmdUtils.get_command("count")
```

```
def on_connection_interrupted(connection, error, **kwargs):
```

```
def on_connection_interrupted(connection, error, **kwargs):
   print("Connection interrupted. error: {}".format(error))

def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print("Connection resumed. return_code: {} session_present: {}".format(return_code, session_present))

    if return_code == mqtt.ConnectReturnCode.ACCEPTED and not session_present:
        print("Session did not persist. Resubscribing to existing topics...")
        resubscribe_future, _ = connection.resubscribe_existing_topics()
        resubscribe_future.add_done_callback(on_resubscribe_complete)

def on_resubscribe_complete(resubscribe_future):
        resubscribe_results = resubscribe_future.result()
        print("Resubscribe results: {}".format(resubscribe_results))

        for topic, qos in resubscribe_results['topics']:
            if qos is None:
                 sys.exit("Server rejected resubscribe to topic: {}".format(topic))

def on_message_received(topic, payload, dup, qos, retain, **kwargs):
    print("Received message from topic '{}': {}".format(topic, payload))
    global received_count
    received_count += 1
    if received_count == cmdUtils.get_command("count"):
        received_all_event.set()

if __name__ == '__main__':
    mqtt_connection = cmdUtils.build_mqtt_connection(on_connection_interrupted, on_connection_resumed)

    print("Connecting to {} with client ID '{}'...".format(
        cmdUtils.get_command(cmdUtils.m_cmd_endpoint), cmdUtils.get_command("client_id")))
    connect_future = mqtt_connection.connect()

    connect_future.result()
    print("Connected!")

    message_count = cmdUtils.get_command("count")
```



```
    message_topic = cmdUtils.get_command(cmdUtils.m_cmd_topic)
    message_string = cmdUtils.get_command(cmdUtils.m_cmd_message)

    # Subscribe
    print("Subscribing to topic '{}'...".format(message_topic))
    subscribe_future, packet_id = mqtt_connection.subscribe(
        topic=message_topic,
        qos=mqtt.QoS.AT_LEAST_ONCE,
        callback=on_message_received)

    subscribe_result = subscribe_future.result()
    print("Subscribed with {}".format(str(subscribe_result['qos'])))

    if message_string:
        if message_count == 0:
            print ("Sending messages until program killed")
        else:
            print ("Sending {} message(s)".format(message_count))

        publish_count = 1
        while (publish_count <= message_count) or (message_count == 0):
            humidity,temperature = Adafruit_DHT.read_retry(22,4)
            temp,pressure,altitude = bmpsensor.readBmp180()
            message = "Temp={0:0.1f}
Humidity={1:0.1f}".format(temperature,humidity,publish_count)
            message = message + " Pressure=" +str(pressure) + " Altitude=" + str(altitude)
            print("Publishing message to topic '{}': {}".format(message_topic, message))
            message_json = json.dumps(message)
            mqtt_connection.publish(
                topic=message_topic,
                payload=message_json,
                qos=mqtt.QoS.AT_LEAST_ONCE)
            time.sleep(1)
            publish_count += 1

    if message_count != 0 and not received_all_event.is_set():
        print("Waiting for all messages to be received...")

    received_all_event.wait()
    print("{} message(s) received.".format(received_count))

   disconnect_future = mqtt_connection.disconnect()
    disconnect_future.result()
```

```
message_topic = cmdUtils.get_command(cmdUtils.m_cmd_topic)
```

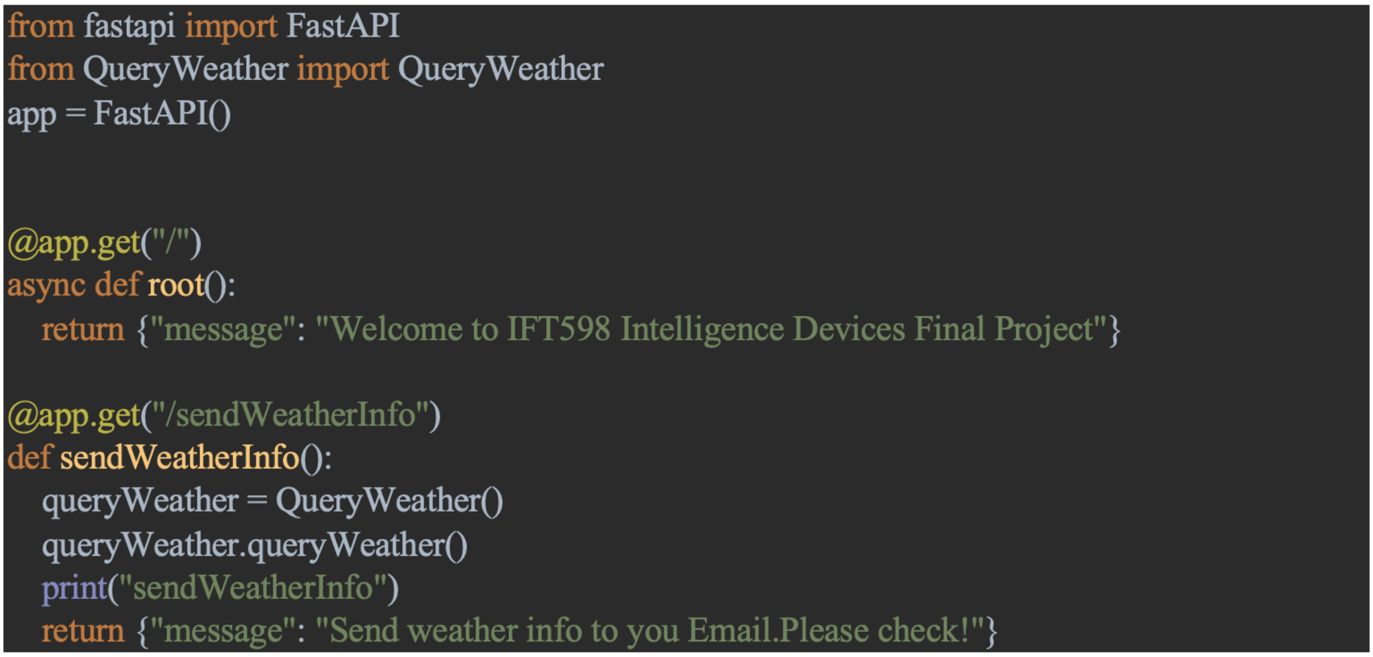
```
message_string = cmdUtils.get_command(cmdUtils.m_cmd_message)

# Subscribe
print("Subscribing to topic '{}'...".format(message_topic))
subscribe_future, packet_id = mqtt_connection.subscribe(
    topic=message_topic,
    qos=mqtt.QoS.AT_LEAST_ONCE,
    callback=on_message_received)

subscribe_result = subscribe_future.result()
print("Subscribed with {}".format(str(subscribe_result['qos'])))

if message_string:
    if message_count == 0:
        print ("Sending messages until program killed")
    else:
        print ("Sending {} message(s)".format(message_count))

   publish_count = 1
   while (publish_count <= message_count) or (message_count == 0):
       humidity, temperature = Adafruit_DHT.read_retry(22,4)
       temp, pressure, altitude = bmpsensor.readBmp180()
       message = "Temp={0:0.1f}
       Humidity={1:0.1f}".format(temperature, humidity, publish_count)
       message = message + " Pressure=" +str(pressure) + " Altitude=" + str(altitude)
       print("Publishing message to topic '{}': {}".format(message_topic, message))
       message_json = json.dumps(message)
       mqtt_connection.publish(
           topic=message_topic,
           payload=message_json,
           qos=mqtt.QoS.AT_LEAST_ONCE)
      time.sleep(1)
      publish_count += 1

if message_count!= 0 and not received_all_event.is_set():
    print("Waiting for all messages to be received...")

received_all_event.wait()
print("{} message(s) received.".format(received_count))

disconnect_future = mqtt_connection.disconnect()
disconnect_future.result()
```

Part of this code is adapted from the AWS IoT Device SDK for Python (v2) repository available at *https://github.com/aws/aws-iot-device-sdk-python-v2, used under the Apache 2.0 License.*

Run Command:

```
python3 weather.py --topic weather --ca_file ~/certs/AmazonRootCA1.pem --cert ~/certs/certificate.pem.crt --key ~/certs/private.pem.key --endpoint a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com
```

Restful server - fastAPI - main.py



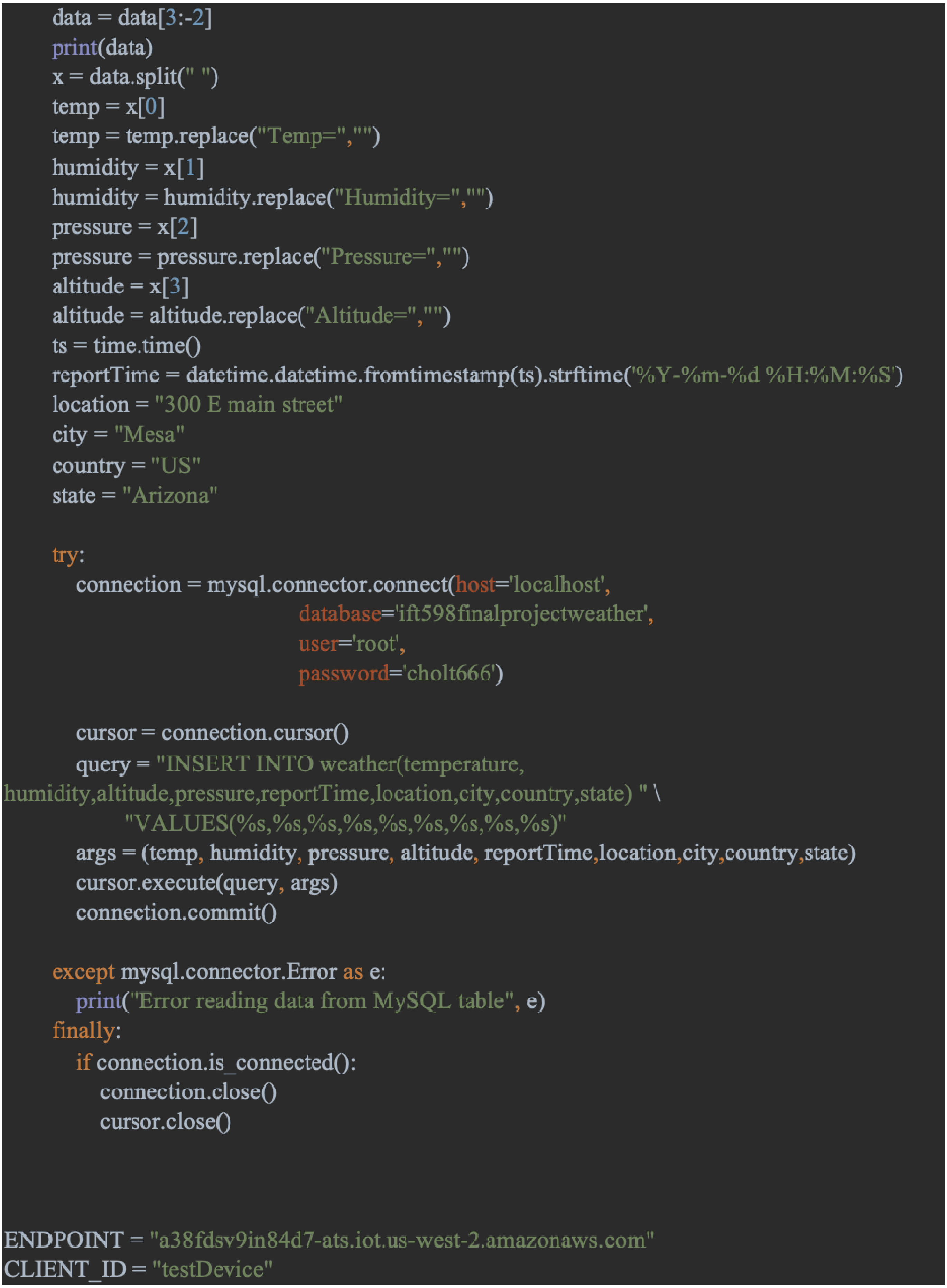
```python
from fastapi import FastAPI
from QueryWeather import QueryWeather
app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Welcome to IFT598 Intelligence Devices Final Project"}

@app.get("/sendWeatherInfo")
def sendWeatherInfo():
    queryWeather = QueryWeather()
    queryWeather.queryWeather()
    print("sendWeatherInfo")
    return {"message": "Send weather info to you Email.Please check!"}
```

```python
from fastapi import FastAPI
from QueryWeather import QueryWeather

app = FastAPI()@app.get("/")
async def root():
    return {"message": "Welcome to IFT598 Intelligence Devices Final Project"}

@app.get("/sendWeatherInfo")
def sendWeatherInfo():
    queryWeather = QueryWeather()
    queryWeather.queryWeather()
    print("sendWeatherInfo")
    return {"message": "Send weather info to you Email.Please check!"}
```

MessagePersistence.py

```
import time
import datetime
import AWSIoTPythonSDK.MQTTLib as AWSIoTPyMQTT
import mysql.connector


class CallbackContainer(object):

    def __init__(self, client):
        self._client = client

    def messagePersistence(self, client, userdata, message):
        data = message.payload
        data = str(data)
```

```
import time
import datetime
import AWSIoTPythonSDK.MQTTLib as AWSIoTPyMQTT
import mysql.connector


class CallbackContainer(object):


    def __init__(self, client):
        self._client = client


    def messagePersistence(self, client, userdata, message):
        data = message.payload
        data = str(data)
```



```
        data = data[3:-2]
        print(data)
        x = data.split(" ")
        temp = x[0]
        temp = temp.replace("Temp=","")
        humidity = x[1]
        humidity = humidity.replace("Humidity=","")
        pressure = x[2]
        pressure = pressure.replace("Pressure=","")
        altitude = x[3]
        altitude = altitude.replace("Altitude=","")
        ts = time.time()
        reportTime = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
        location = "300 E main street"
        city = "Mesa"
        country = "US"
        state = "Arizona"

        try:
            connection = mysql.connector.connect(host='localhost',
                                                 database='ift598finalprojectweather',
                                                 user='root',
                                                 password='cholt666')

            cursor = connection.cursor()
            query = "INSERT INTO weather(temperature,
humidity,altitude,pressure,reportTime,location,city,country,state) " \
                    "VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s)"
            args = (temp, humidity, pressure, altitude, reportTime,location,city,country,state)
            cursor.execute(query, args)
            connection.commit()

        except mysql.connector.Error as e:
            print("Error reading data from MySQL table", e)
        finally:
            if connection.is_connected():
                connection.close()
                cursor.close()



ENDPOINT = "a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com"
CLIENT_ID = "testDevice"
```

```
    data = data[3:-2]
```

```
        data = data[3:-2]
        print(data)
        x = data.split(" ")
        temp = x[0]
        temp = temp.replace("Temp=","")
        humidity = x[1]
        humidity = humidity.replace("Humidity=","")
        pressure = x[2]
        pressure = pressure.replace("Pressure=","")
        altitude = x[3]
        altitude = altitude.replace("Altitude=","")
        ts = time.time()
        reportTime = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
        location = "300 E main street"
        city = "Mesa"
        country = "US"
        state = "Arizona"

        try:
            connection = mysql.connector.connect(host='localhost',
                database='ift598finalprojectweather',
                user='root',
                password='cholt666')

            cursor = connection.cursor()
            query = "INSERT INTO weather(temperature, humidity, altitude, pressure, reportTime, location, city, country, state) " \
                 "VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s)"
             args = (temp, humidity, pressure, altitude, reportTime, location, city, country, state)
             cursor.execute(query, args)
             connection.commit()

       except mysql.connector.Error as e:
           print("Error reading data from MySQL table", e)
       finally:
           if connection.is_connected():
               connection.close()
               cursor.close()

ENDPOINT = "a38fdsv9in84d7-ats.iot.us-west-2.amazonaws.com"
CLIENT_ID = “testDevice”
```

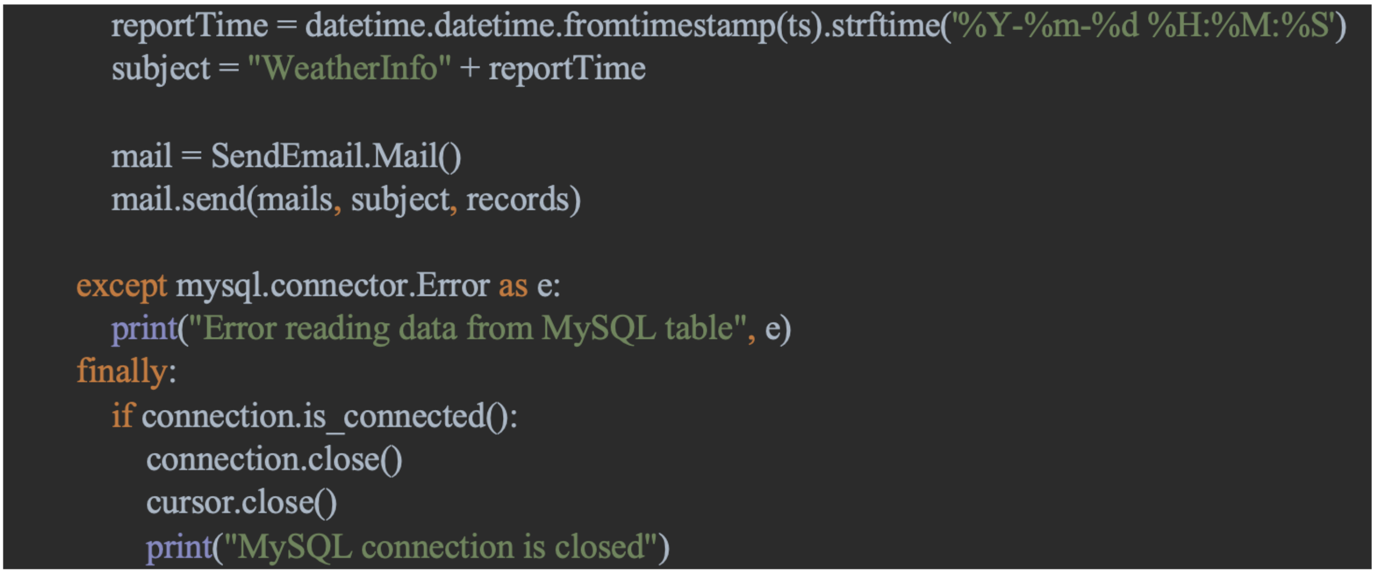
```
PATH_TO_CERTIFICATE = "certificate.pem.crt"
PATH_TO_PRIVATE_KEY = "private.pem.key"
PATH_TO_AMAZON_ROOT_CA_1 = "AmazonRootCA1.pem"
TOPIC = "weather"
RANGE = 20

myAWSIoTMQTTClient = AWSIoTPyMQTT.AWSIoTMQTTClient(CLIENT_ID)
myAWSIoTMQTTClient.configureEndpoint(ENDPOINT, 8883)
myAWSIoTMQTTClient.configureCredentials(PATH_TO_AMAZON_ROOT_CA_1,
PATH_TO_PRIVATE_KEY, PATH_TO_CERTIFICATE)

myAWSIoTMQTTClient.connect()
myCallbackContainer = CallbackContainer(myAWSIoTMQTTClient)
myAWSIoTMQTTClient.subscribe(TOPIC, 1, myCallbackContainer.messagePersistence);
time.sleep(10)

myAWSIoTMQTTClient.disconnect()
```

```
PATH_TO_CERTIFICATE = "certificate.pem.crt"
PATH_TO_PRIVATE_KEY = "private.pem.key"
PATH_TO_AMAZON_ROOT_CA_1 = "AmazonRootCA1.pem"
TOPIC = "weather"
RANGE = 20

myAWSIoTMQTTClient = AWSIoTPyMQTT.AWSIoTMQTTClient(CLIENT_ID)
myAWSIoTMQTTClient.configureEndpoint(ENDPOINT, 8883)
myAWSIoTMQTTClient.configureCredentials(PATH_TO_AMAZON_ROOT_CA_1, PATH_TO_PRIVATE_KEY, PATH_TO_CERTIFICATE)

myAWSIoTMQTTClient.connect()
myCallbackContainer = CallbackContainer(myAWSIoTMQTTClient)
myAWSIoTMQTTClient.subscribe(TOPIC, 1, myCallbackContainer.messagePersistence);
time.sleep(10)

myAWSIoTMQTTClient.disconnect()
```

QueryWeather.py

```
import mysql.connector
import SendEmail
import datetime
import time


class QueryWeather:

    def queryWeather(self):
        try:
            connection = mysql.connector.connect(host='localhost',
                                                 database='ift598finalprojectweather',
                                                 user='root',
                                                 password='cholt666')

            sql_select_Query = "SELECT * FROM weather"
            cursor = connection.cursor()
            cursor.execute(sql_select_Query)
            records = cursor.fetchall()

            mails = input("Enter emails: ").split()
            ts = time.time()
```

```
import mysql.connector
import SendEmail
import datetime
import time

class QueryWeather:

    def queryWeather(self):

        try:
            connection = mysql.connector.connect(host='localhost',
                database='ift598finalprojectweather',
                user='root',
                password='cholt666')

           sql_select_Query = "SELECT * FROM weather"
           cursor = connection.cursor()
           cursor.execute(sql_select_Query)
           records = cursor.fetchall()

           mails = input("Enter emails: ").split()
           ts = time.time()
```



```
        reportTime = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
        subject = "WeatherInfo" + reportTime

        mail = SendEmail.Mail()
        mail.send(mails, subject, records)

    except mysql.connector.Error as e:
        print("Error reading data from MySQL table", e)
    finally:
        if connection.is_connected():
            connection.close()
            cursor.close()
            print("MySQL connection is closed")
```

```
        reportTime = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
        subject = "WeatherInfo" + reportTime

        mail = SendEmail.Mail()
        mail.send(mails, subject, records)

    except mysql.connector.Error as e:
        print("Error reading data from MySQL table", e)
    finally:
        if connection.is_connected():
            connection.close()
            cursor.close()
            print("MySQL connection is closed")
```

SendEmail.py

```
import smtplib, ssl


class Mail:

    def __init__(self):
        self.port = 465
        self.smtp_server_domain_name = "smtp.gmail.com"
        self.sender_mail = "wj4507657@gmail.com"
        self.password = ""

    def send(self, emails, subject, content):
        ssl_context = ssl.create_default_context()
        service = smtplib.SMTP_SSL(self.smtp_server_domain_name, self.port,
context=ssl_context)
        service.login(self.sender_mail, self.password)

        for email in emails:
            result = service.sendmail(self.sender_mail, email, f"Subject: {subject}\n{content}")

        service.quit()
```

```
import smtplib, ssl


class Mail:


    def __init__(self):

        self.port = 465

        self.smtp_server_domain_name = "smtp.gmail.com"

        self.sender_mail = "wj4507657@gmail.com"

        self.password = ""


    def send(self, emails, subject, content):

        ssl_context = ssl.create_default_context()

        service = smtplib.SMTP_SSL(self.smtp_server_domain_name, self.port, context=ssl_context)

        service.login(self.sender_mail, self.password)


        for email in emails:

            result = service.sendmail(self.sender_mail, email, f"Subject: {subject}\n{content}")


        service.quit()
```

Reference

- Bahga, A., & Madisetti, V. (2014). Internet of Things: A hands-on approach. Vpt.